



Calhoun: The NPS Institutional Archive
DSpace Repository

Theses and Dissertations

1. Thesis and Dissertation Collection, all items

2006-12

Traffic profiling of wireless sensor networks

Kirykos, Georgios

Monterey, California. Naval Postgraduate School

<http://hdl.handle.net/10945/2442>

Downloaded from NPS Archive: Calhoun



Calhoun is the Naval Postgraduate School's public access digital repository for research materials and institutional publications created by the NPS community. Calhoun is named for Professor of Mathematics Guy K. Calhoun, NPS's first appointed -- and published -- scholarly author.

Dudley Knox Library / Naval Postgraduate School
411 Dyer Road / 1 University Circle
Monterey, California USA 93943

<http://www.nps.edu/library>



NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

TRAFFIC PROFILING OF WIRELESS SENSOR NETWORKS

by

Georgios Kirykos

December 2006

Thesis Advisor:
Second Reader:

John McEachen
Murali Tummala

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			<i>Form Approved OMB No. 0704-0188</i>	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.				
1. AGENCY USE ONLY (Leave blank)		2. REPORT DATE December 2006	3. REPORT TYPE AND DATES COVERED Master's Thesis	
4. TITLE AND SUBTITLE: Traffic Profiling in Wireless Sensor Networks			5. FUNDING NUMBERS	
6. AUTHOR(S) Georgios Kirykos				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A			10. SPONSORING/MONITORING AGENCY REPORT NUMBER	
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.				
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited			12b. DISTRIBUTION CODE	
13. ABSTRACT (maximum 200 words) Network security is vital in wireless networks that are widely used today. We desire wireless networks that maintain a high degree of confidentiality, integrity, and availability. Wireless sensor networks pose unique challenges and limitations to the traditional schemes, which are used in the other wireless networks for security protection, and are due mainly to the increased vulnerability of physical attacks, energy and communication limitations. This thesis introduces the foundations of a network and anomaly-based Intrusion Detection System (IDS) tool, including both hardware and software components, that can be used for traffic profiling and monitoring of a wireless sensor network. The work demonstrates how the IDS should capture and store traffic and use this information to create traffic profiles and baselines for normal traffic behavior. Then it describes how these baselines can be used to generate alerts based on traffic variations that imply possible attacks. Profiles on typical implementations of wireless sensor networks were observed and analyzed. Finally, initial indications from basic analysis of wireless sensor network traffic demonstrated a high degree of self-similarity.				
14. SUBJECT TERMS Wireless Sensor Networks, Intrusion Detection System (IDS), Self similarity, Packet Sniffer			15. NUMBER OF PAGES 87	
			16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL	

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. Z39-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

TRAFFIC PROFILING OF WIRELESS SENSOR NETWORKS

Georgios Kirykos
Lieutenant, Hellenic Navy
B.S., Hellenic Naval Academy, 1996

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

**NAVAL POSTGRADUATE SCHOOL
December 2006**

Author: Georgios Kirykos

Approved by: John C. McEachen
Thesis Advisor

Murali Tummala
Second Reader

Jeffrey B. Knorr
Chairman, Department of Electrical and Computer Engineering

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Network security is vital in wireless networks that are widely used today. We desire wireless networks that maintain a high degree of confidentiality, integrity, and availability. Wireless sensor networks pose unique challenges and limitations to the traditional schemes, which are used in the other wireless networks for security protection, and are due mainly to the increased vulnerability of physical attacks, energy and communication limitations. This thesis introduces the foundations of a network and anomaly-based Intrusion Detection System (IDS) tool, including both hardware and software components, that can be used for traffic profiling and monitoring of a wireless sensor network. The work demonstrates how the IDS should capture and store traffic and use this information to create traffic profiles and baselines for normal traffic behavior. Then it describes how these baselines can be used to generate alerts based on traffic variations that imply possible attacks. Profiles on typical implementations of wireless sensor networks were observed and analyzed. Finally, initial indications from basic analysis of wireless sensor network traffic demonstrated a high degree of self-similarity.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

I.	INTRODUCTION.....	1
A.	BACKGROUND	1
B.	OBJECTIVES	1
C.	RELATED WORK	2
D.	THESIS ORGANIZATION.....	2
II.	BACKGROUND THEORY FOR UNATTENDED SENSOR NETWORKS.....	5
A.	APPLICATIONS	6
1.	Military Applications.....	6
2.	Environmental Detection and Monitoring.....	7
3.	Civil Engineering and Home Intelligence	8
4.	Health Monitoring	8
B.	ELEMENTS OF WIRELESS SENSOR NETWORKS	8
C.	TECHNOLOGY COMPONENT EVOLUTION.....	9
1.	Digital Circuitry	11
2.	Wireless Communications.....	11
3.	Microelectromechanical Systems (MEMS)	12
D.	SUMMARY	14
III.	SENSOR MOTE TYPES	15
A.	MOTE TYPES AND DESCRIPTION	15
1.	Micaz mote.....	15
2.	Telos Rev B Mote	16
3.	MIB 510 Programming Platform	17
4.	MTS 310 Multipurpose Sensor Board	18
a.	<i>Microphone and Tone Detector.....</i>	<i>18</i>
b.	<i>Temperature Detector (Panasonic ERT-J1VR103J).....</i>	<i>18</i>
c.	<i>Two-axis Accelerometer (ADI ADXL202)</i>	<i>18</i>
d.	<i>Two-axis Magnetometer (HMC1002)</i>	<i>18</i>
e.	<i>Sounder.....</i>	<i>18</i>
B.	EXPERIMENT LAYOUT	19
C.	SUMMARY	20
IV.	SENSOR NETWORK TRAFFIC TYPES	21
A.	802.15.4 AND ZIGBEE ARCHITECTURE.....	21
B.	TINYOS/XMESH PACKET STRUCTURE AND ANALYSIS	23
C.	SUMMARY	28
V.	TRAFFIC CAPTURE SNIFFER DESIGN	31
A.	SNIFFER ANATOMY	31
B.	DESCRIPTION OF JAVA CLASSES.....	33
C.	OUTPUT	34
D.	SUMMARY	36
VI.	TRAFFIC CATEGORIZATION FRAMEWORK	37

A.	TRAFFIC STATISTICS	38
1.	Direct Motes to Base Connection.....	38
2.	Parent-Child Network Deployment.....	40
B.	ESTIMATING SELF SIMILARITY IN WSN	45
C.	SUMMARY	49
VII.	CONCLUSIONS AND FUTURE WORK.....	51
A.	CONCLUSIONS	51
B.	FUTURE WORK.....	52
APPENDIX A.	55
A.	SNIFFER JAVA CLASS.....	55
B.	ACTIVE MESSAGE JAVA CLASS.....	60
C.	OUTPUTTOCSV JAVA CLASS.....	61
D.	DISPLAYWINDOW JAVA CLASS	61
APPENDIX B.	63
LIST OF REFERENCES	65
INITIAL DISTRIBUTION LIST	67

LIST OF FIGURES

Figure 1.	Basic elements of a Wireless Sensor Network (From Ref. 1).	9
Figure 2.	Illustration of decreasing in size of computational devices vs. time (From Ref. 1).	10
Figure 3.	Solar powered, bi-directional communications & sensing (acceleration, ambient light) (From Ref. 8).	10
Figure 4.	Gear set produced using MEMS compared with a mite (From Ref. 8).	14
Figure 5.	Micaz hardware description (From Ref. 1).	15
Figure 6.	TelosB hardware description (From Ref. 1).	16
Figure 7.	MIB510 Serial Interface Programming board (From Ref. 1).	18
Figure 8.	MTS 310 Multipurpose sensor board (From Ref. 1).	19
Figure 9.	Experiment layout with a base station on the top right.	19
Figure 10.	IEEE 802.15.4 and Zigbee Layer Architecture (From Ref. 1).	21
Figure 11.	Xmesh and Zigbee protocol layering share the same MAC and PHY lower layer (From Ref. 1).	22
Figure 12.	TinyOS Physical Message Packet (From Ref. 1).	23
Figure 13.	TinyOS Packet Structure (From Ref. 1).	24
Figure 14.	Sniffer application block diagram.	32
Figure 15.	Illustration of the displayWindow java program; a GUI for traffic monitoring.	35
Figure 16.	Illustration of the CSV file format.	35
Figure 17.	Traffic percentage statistics in accordance to Active Message type.	39
Figure 18.	Traffic percentage statistics in accordance to Active Message type for the Parent-Child setup.	41
Figure 19.	Traffic percentage statistics comparison in accordance to Active Message type for Direct Base-Motes communication and Parent-Child setup.	41
Figure 20.	Number of packets vs. payload length in direct communication scenario.	44
Figure 21.	Number of packets vs. Payload Length in Parent-Child communication scenario.	45
Figure 22.	Logarithm of Variance Payload Length vs. Aggregated Average Payload Length Plot for the direct scenario.	47
Figure 23.	Logarithm of the Variance of packet Interarrival Time versus Aggregated Time plot for the direct to base communication scenario.	48
Figure 24.	Logarithm of variance of packet Payload Length versus Aggregated Average Packet Length plot for the Parent-Child communication scenario.	48
Figure 25.	Logarithm of the Variance of packet Interarrival Time versus Aggregated Time plot for the Parent-Child communication scenario.	49

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

Table 1.	Chronological evolution of motes from different vendors in terms of size, weight, power supply, topology, deployment methods. (From Ref. 1)	12
Table 2.	Micaz performance characteristics. (From Ref. 1)	16
Table 3.	TelosB performance characteristics. (From Ref. 1).....	17
Table 4.	Identification Number for Active Message Types (From Ref. 12).....	26
Table 5.	Summary of the Base and Motes messages properties and differences.....	28
Table 6.	Traffic characteristics for all Active Message type on direct scenario.....	40
Table 7.	Traffic characteristics according to active message types for the parent-child radio communication scheme.	43

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

I would like to thank Professor John McEachen who has been my thesis supervisor and mentor. Your insight is absolutely invaluable and your knowledge in Electrical and Computer Engineering is immeasurable.

Thank you, to my second reader, Professor Murali Tummala for bringing your unique talent to this project. A strong foundation of knowledge was built through your teaching.

Thank you, to Lt Timothy Zane, USN, for your guidance and help in computer programming and computer science related materials.

Thank you to my friends Nikolaos Alchazidis, Stefanos Filtikakis, and Evanna Stefanakis for your help and support.

Lastly, and most importantly, I wish to thank my parents, Vassilios Kirykos and Irene Kirykos, who have unfailingly given me constant support throughout my life. I thank you for always believing in me and your endless knowledge and guidance are the reason I continue to press on in my career. To you, I dedicate this thesis.

THIS PAGE INTENTIONALLY LEFT BLANK

EXECUTIVE SUMMARY

Human need has always been the factor that has driven all technologies and applications. Today with the wide spread of internet and wireless communications, the human need for real-time sensing of the environment has emerged which has led to the development of wireless sensor networks. Although other wireless technologies have been implemented for many years and many issues have arisen and been solved, these networks pose unique challenges. Physical vulnerability, energy and communication constraints distinguish sensor networks from the IEEE 802.11 family wireless networks. These dissimilarities impose extra security vulnerabilities that the existing protocols and tools cannot address effectively.

One way of addressing network security is through intrusion detection. A network-based anomaly intrusion detection system requires a packet sniffer. A packet sniffer is a device composed of hardware and software components that can be used to capture and store the exchanged data within a network. The procedure that should follow consists of two parts. The first part deals with capturing the data and analyzing the exchanged traffic to identify traffic characteristics. This traffic profiling and categorization can be used as a baseline of normal behavior. The second part simply compares real-time traffic characteristics with normal traffic and checks if differences would apply. The major advantage of this technique is that it can identify all possible attacks (not only the known ones).

The layered architecture for the motes that were used follows the two lower layers of the OSI model where the physical and Medium Access Control (MAC) layers adhere to the IEEE 802.15.4 standard. This is especially true for the physical layer, which was implemented into hardware within a CC2400 Chipcon transceiver. The upper layers were modeled by Crossbow Inc. applications, such as Xmesh and Surge, and complied with the Zigbee alliance higher layer architecture. The operating system that handles the intercommunication between hardware and software components was TinyOS. The traffic profiling was based on the MAC layer packet structure and

more specifically upon the Active Message index. This index is used by Xmesh and is used to handle the incoming packets from the MAC layer to the appropriate application. It is very similar to the TCP port number.

In this thesis, a packet sniffer consisted of a java program *sniffer* and a hardware sniffing component. Specifically, a TelosB mote was used to capture the traffic between three motes. Two basic deployment topologies for the motes were used. In the first case, the motes were in line of sight and within radio distance of the base station. In the second scenario, the one mote was put out of radio distance of the base mote so that its data would have to be relayed by the other mote, creating a “parent-child” relationship.

All data were stored on a laptop computer that ran the packet sniffer device in a comma separated value (CSV) format, which was then imported into an excel spreadsheet for statistical analysis. The analysis was based on grouping the packets in terms of Active Message index and then comparing the destination and originating address, interarrival times, and packet payload length. This process revealed the following.

Each traffic type occupied a different percentage of the total traffic. It was found that the data traffic encompassed 71%, health and routing traffic 23%, and finally acknowledgement 6% of the total traffic.

Examination of the packet length and the interarrival times of the packets in different time aggregations, in a variance-time plot on a logarithmic scale, indicated that the traffic is self-similar. That was also witnessed by visual examination of the traffic, where each packet type was clustered. This means that every packet is largely dependent on the previous packets received, and upon reception of a specific message type, the same packet type is more likely to be received.

A comparison of the different packet type payload lengths and interarrival times revealed some discrepancies with the protocol specifications with very large variations, a fact which requires further investigation.

The payload lengths observed extended from 11 to 27 bytes and never exceeded 29 bytes, which is the maximum packet length for the protocol used. Different packet types used specific payload lengths.

To summarize, the traffic categorization, according to Active Message type, held enough information to characterize the traffic, the java program *Sniffer* can be used as a network-based, anomaly-based, intrusion detection system. The overall traffic demonstrated long range dependency since was found to be self-similar with high Hurst parameter values ($H > 0.98$).

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. BACKGROUND

Wireless Sensor Networks are becoming important for many commercial and military applications where real time information is needed and there is lack of Information Technology (IT) infrastructure. These networks consist of a number of sensors, usually tens to thousands, and are spread throughout a specified geographical area. Some examples of these networks are the following [1]:

1. Military networks, designed to detect and gain information about an enemy's location and movement, weapon explosions, and chemical, biological, radiological and nuclear attacks.

2. Wireless sensor networks that detect and monitor environmental changes across plains, forests and oceans. Agricultural operations are a good example in this category.

3. Infrastructure monitoring wireless sensor networks, which monitor vehicle traffic on highways, the congested parts of cities and other similar applications. An extension in this area is the security monitoring of large public areas, such as malls and parking garages, and control and monitoring of machines in industries.

In such environments as briefly described above, IT infrastructure is not present or is very poor, and cabling is prohibited due to cost, time or access. The most inexpensive, quick and reliable way of information collection can be provided by the deployment of a wireless sensor network.

B. OBJECTIVES

The objective of this thesis is to develop a tool that would allow investigating traffic characteristics of unattended wireless sensor networks consisting of motes interacting in a traditional setting. More specifically, the proposed tool allows the identification of different packet types that include control, management, routing and user data, and categorizes traffic in such way to allow for the identification of anomalies and variations in traffic patterns that could be used in an intrusion detection system.

C. RELATED WORK

All packet-based networks, both wired and wireless, are vulnerable to attacks, a fact that provides a very good background of what we should expect in a wireless sensor network and serves as very good point for further research. One of the tools that is widely used today, as a traffic analysis, diagnosis and attack tool, is a packet sniffer.

For the family of IEEE 802.11 wireless networks, widely used packet sniffers are Ethereal [2], Tcpdump [3] and others. Those programs are widely used to analyze network traffic, to detect network intrusion attacks, to provide network statistics, and/or to capture sensitive information from other networks.

A packet sniffer for TinyOS-based wireless sensor networks was designed by Hong-Siang Teo [4] and was used to validate some of the security properties that exist in such a sensor network. This packet sniffer was modified by this thesis, with the addition of several components that served the functionalities needed for traffic analysis and profiling, according to specific components of the Medium Access Control (MAC) header and payload.

D. THESIS ORGANIZATION

Chapter I contains an introduction to the thesis and aims to familiarize the reader with wireless sensor network applications and related work in the literature. Chapter II provides background information and a discussion of the evolution of wireless sensor networks as well as their components. It also discusses why these networks are becoming very popular nowadays, increasing their share in the market. In Chapter III, a detailed description of the hardware components and their specifications that were used in this experiment is provided. This chapter closes with a layout of the experiment, describing how the wireless sensor network was deployed, what hardware components were used, and how the traffic was captured.

Chapter IV deals with the software components. It provides an analysis of the protocol used to handle the information exchange in this network and emphasizes the components which this thesis is going to use for the categorization of the data traffic. Chapter V provides a description of the software used for the traffic capturing, emphasizing the three programs that were written for this purpose and providing a

detailed analysis of their functionality. The detail of the program code is printed in Appendix A. The analysis and profiling of the traffic follows in Chapter VI, and the thesis ends with Chapter VII, where conclusions and future work are discussed.

THIS PAGE INTENTIONALLY LEFT BLANK

II. BACKGROUND THEORY FOR UNATTENDED SENSOR NETWORKS

It is evident that the research of commercial and military Wireless Sensor Networks (WSN) is growing exponentially. This fact can be manifested by the increasing number of searches conducted through Internet sources, such as *Google*. In August 2003 there were 8,000 hits for WSN while today the number of hits exceeds 26,000 [5]. There has also been an increase in the amount of annual workshops, some of which include IPSN (Information Processing in Sensor Networks), SenSys, EWSN (European Workshop on Wireless Networks and Applications) and other conference sessions dedicated to this subject, such as ISIT, ICC, INFOCOM [5], and the IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks (SECON) [6]. Many vendors today offer a variety of applications and motes which are able to accomplish almost any kind of commercial and military need for remote, real-time applications and more are to come.

Compared to the use of a few, large, expensive but highly accurate sensors, it is more cost efficient to deploy a large number of inexpensive sensors that offer the advantage of a smaller total system expenditure with much higher spatial resolution, higher robustness, uniform coverage, and ease of deployment, anywhere and anytime with reduced energy consumption [5]. These inexpensive sensors are lower in cost when compared to microprocessor systems, which is due to advancements in technology that have permitted the manufacture of such low cost motes.

As mentioned above, the small sensor motes achieve higher spatial resolution, in comparison to complex microprocessor systems within a similar-sized geographical area, because the sensing of the environment is done by a large number of motes instead of just one. Appropriate algorithms at the receiver use correlation across the received data to enhance the received signal and output more accurate reading/results. Higher robustness is achieved by dense deployment of the motes in a geographical area where a number of nodes may fail but the sensor system is still able to read and transmit acceptable information. This is another advantage over a single microprocessor system, which if it fails it will cause the loss of its functionality in a given area. Also, uniform coverage can

easily be achieved with the appropriate distribution of the sensors in the area of interest, in comparison to the single microprocessor sensor, where it is limited to placement in the center of the interest area. One final advantage of the lower cost system is that it can be placed anywhere and anytime since it does not require human attention or the need of physical presence in the area of interest, which can be hazardous or unreachable. The system can be distributed by airborne vehicles, such as airplanes or helicopters [5].

The uniqueness of an unattended wireless sensor network is that since human presence is not necessary the system can automatically perform node setup and network boot-up. The network must be able to make dynamic adjustments in order to be able to handle the changes in the environment and to itself. That means it should also be able to optimize its functioning, be able to fine tune work flow to achieve predetermined goals and be able to recover from routine or extraordinary events that might cause it to malfunction. The network must also have the capability to discover problems, such as uncovered areas due to mote(s) malfunctions and to find alternate resources and routes to keep it functioning properly and smoothly. Finally, it must detect, identify and protect itself against various types of attacks in order to maintain overall system security and integrity.

This thesis will build tools and create profiles that can be used by the network administrator, to gather and monitor the traffic of an unattended wireless sensor network and identify the various attacks as mentioned in the paragraph above.

A. APPLICATIONS

This paragraph will provide, in more detail than in the introduction, some applications areas of the wireless sensor networks, used today, as well as some experimental projects that have not been utilized yet but most likely will be seen in the near future.

1. Military Applications

Wireless Sensor Networks (WSN) are becoming a very important part of the military command, control, communications, computing, intelligence, surveillance, reconnaissance and targeting systems. In the battlefield, the tendency of the targets is that they will become smaller and less detectable, and that they will have higher mobility and the ability to move in any kind of terrain and environment [5], [7]. It seems that a

WSN deployed in high density environment where the potential target is or is expected to be proven an excellent source of information for the detection of the hostile forces, remote sensing of the release of nuclear, biological and chemical weapons, and in a variety of other remote sensing operations. Also, commanders can monitor the status and location of their troops, weapons and supplies in order to improve military command, control and communications [5], [7] as well as smart minefields that can sense the intruders and react appropriately.

2. Environmental Detection and Monitoring

The spread of tiny, inexpensive nodes in a wide geographical area can be used for many types of applications such as flood detection, air and sewage monitoring, climate control in very large buildings, soil composition in agriculture, forest fire detection, and geophysical studies as well as to survey the plant and animal population.

Also, appropriate sensors can be used for crop and livestock monitoring and management in a very large agricultural area, where physical monitoring is mandatory but time consuming, or sometimes inaccessible. In such situations, with the deployment of the appropriate sensor network, soil and plant elements can be monitored, and based on the resultant readings the fertilizer concentration needed can be calculated with accuracy and in real time.

In the field of the seismic activity detection, a WSN that consists of accelerometers can provide a much finer scale than having a single sensor for a very large area since nodes can be deployed with a very high density within the same area of interest.

Also, wireless sensor networks can assist in the monitoring of fresh water quality in a remote area where sampling is difficult, or even impossible. Several sensors can sample and analyze the water, and then transmit the results accordingly.

Finally, in the area of disaster detection and prevention, such as forest fires and floods, a deployed wireless sensor network can monitor temperature and water levels, and thus provide an early warning [5], [7].

3. Civil Engineering and Home Intelligence

Monitoring of structures like bridges in order to detect and to warn of structural weakness, or the reaction of tall buildings to wind and earthquakes are some of the civil engineering applications of WSNs [5], [7].

In the area of home intelligence, WSN applications can provide smoke detection in order to prevent fires and their spread; automate the reading of gas, water and electricity reading/levels; and facilitate safety monitoring through remote surveillance to detect intruders.

4. Health Monitoring

WSNs can record physiological data, such as body temperature, blood pressure, pulse and other readings, which can be sensed and transmitted automatically to a physician or doctor. In the same manner additional sensors can analyze the blood stream periodically and provide 24-hour coverage of personal health [5], [7].

Remote virus monitoring can be performed in an area that has been infected, and it is not yet safe to humans. These readings can be used by a virus control center in order to determine the spread of the virus and to assist in making decisions, such as when to send appropriate medical support.

B. ELEMENTS OF WIRELESS SENSOR NETWORKS

Wireless Sensor Networks consist of ten to thousands of ultra-small fully autonomous computing and communicating devices with very restricted energy and computing capabilities. These devices cooperate to quickly and efficiently accomplish a large sensing task in a remote area where no cabling is present or human presence is restricted due to several conditions [5].

As illustrated in Figure 1 below, a “Sensor Node” consists of two basic components: a sensor and a mote. The sensor part can be defined as the component that is able to detect/sense physical phenomena such as heat, light, sound, acoustic pressure, acceleration and vibration, and it is usually a cluster of sensors placed on a board. The sensor then transforms the readings into an electrical signal and transmits it to the processor, which is usually on the same board, for further processing [1]. These devices will be better detailed, examined and presented in the next chapter where an analysis of

the sensors used for the experiment is provided. The mote component consists of a microcontroller with a sensor application and networking software, a low power radio transceiver and a power component which can be either a battery or a solar plate [1].

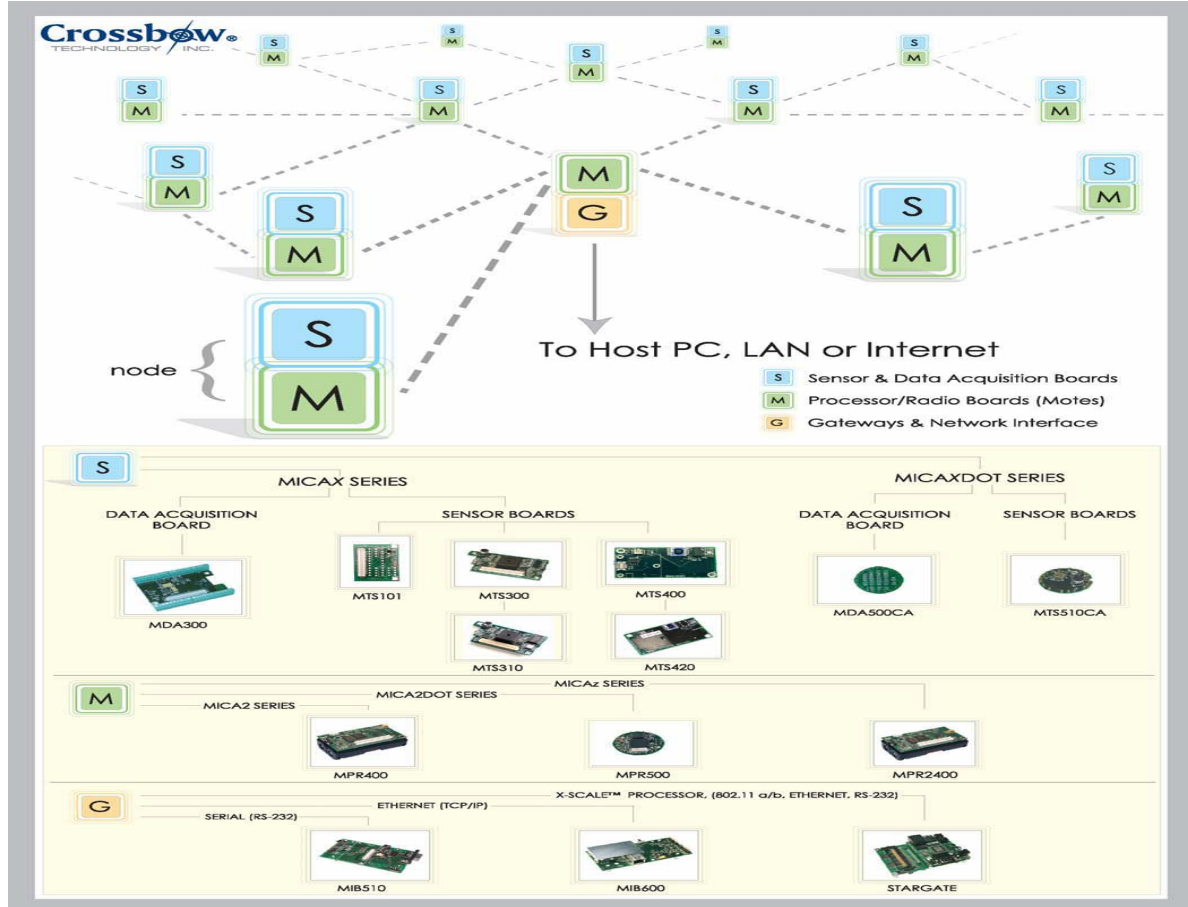


Figure 1. Basic elements of a Wireless Sensor Network (From Ref. 1).

C. TECHNOLOGY COMPONENT EVOLUTION

The evolution of each of the elements of a wireless sensor network has enabled us today to be able to manufacture very small motes, that are able to perform many tasks and to be deployed almost anywhere having the ability to be self-managed, self-organized, self-healing and self-diagnostic [5]. These technologies can be broken down into three major categories: digital circuitry, wireless communications, and microelectromechanical systems (MEMS).

It was not long ago when the first computers were quite large in size especially in comparison to the size of a human, as can be seen starting from upper left corner of

pictures in Figure 2 and moving towards the lower right corner [1]. These sizable pieces of machinery used to occupy an entire room. Today extensive advancement in technology has enabled the manufacture of Personal Computers (PC) that can fit on a small desk, laptops which are easily transportable in small bags, PDA's – which are also known as *Palm Pilots* because of their size – to fit in one's hand, and motes that are equivalent to the size a coin, if not smaller.



Figure 2. Illustration of decreasing in size of computational devices vs. time (From Ref. 1).

Figure 3 demonstrates the size of tiny motes in comparison to a penny [8].



Figure 3. Solar powered, bi-directional communications & sensing (acceleration, ambient light) (From Ref. 8).

1. Digital Circuitry

With digital circuitry discovery and evolution [5], electronic devices have become extremely fast enabling them to perform millions of operations per second; tasks that would have taken humans years and years of effort, all the while performed with no mistakes. But not only the calculation power has increased dramatically - and of course it will continue to increase - the size of the circuit required became extremely small. Today there are devices that are built much larger than they need to be so that humans can hold them in their hand to operate them.

2. Wireless Communications

The evolution in wireless communications using MEMS that is briefly described in the next paragraph assisted in the manufacture of very small transmitters and receivers. The use of different and multiple coding schemes permitted an increase in the rate of transmitted data within the same bandwidth, as well as the energy per single bit that is transmitted.

Table 1 shown below, gives a good illustration of how the motes have been reduced in size, as well as some other characteristics like weight, topology and power supply modules from the different vendors. The illustration includes information from as early as 1980, what we have today and peaks into the future to show what is expected in the year 2010.


	Yesterday (1980's – 1990's)	Today (2000 – 2003)	Tomorrow (2010)
Manufacturer	Custom contractors, e.g., for TRSS	Commercial: Crossbow Technology, Inc. Sensoria Corp., Ember Corp.	Dust, Inc. and others to be formed
Size	Large shoe box and up	Pack of cards to small shoe box	Dust particle
Weight	Kilograms	Grams	Negligible
Node architecture	Separate sensing, processing and communication	Integrated sensing, processing and communication	Integrated sensing, processing and communication
Topology	Point-to-point, star	Client server, peer to peer	Peer to peer
Power supply lifetime	Large batteries; hours, days and longer	AA batteries; days to weeks	Solar, months to years
Deployment	Vehicle-placed or air-drop single sensors	Hand-emplaced	Embedded, "sprinkled" left-behind
			
<div>TRSS Node</div> <div>Crossbow</div> <div>Ember</div> <div>Sensoria</div> <div>Dust, Inc.</div>			

Table 1. Chronological evolution of motes from different vendors in terms of size, weight, power supply, topology, deployment methods. (From Ref. 1)

3. Microelectromechanical Systems (MEMS)

Micro-Electro-Mechanical Systems (MEMS) are the integration of mechanical elements, sensors, actuators and electronics on a common silicon, polymer or metal [7] substrate through microfabrication technology [8], [9], [10].

While the electronics are fabricated using integrated circuit (IC) process sequences (e.g., CMOS, Bipolar, or BICMOS processes), the micromechanical components are fabricated using compatible "micromachining" processes that selectively etch away parts of the silicon wafer or add new structural layers to form the mechanical and electromechanical devices [8], [9], [10] .

MEMS bring together silicon-based microelectronics with micromachining technology, making it possible to have a complete systems-on-a-chip. MEMS is an enabling technology allowing the development of smart products, augmenting the computational ability of microelectronics with the perception and control capabilities of microsensors and microactuators, and expanding the space of possible designs and applications [8], [9], [10].

Microelectronic integrated circuits can be thought of as the "brains" of a system and MEMS augments this decision-making capability with "eyes" and "arms," to allow microsystems to sense and control the environment. Sensors gather information from the environment through measuring mechanical, thermal, biological, chemical, optical and magnetic phenomena. The electronics then process the information derived from the sensors, and through some decision making capability, direct the actuators to respond by moving, positioning, regulating, pumping and filtering, thereby controlling the environment for some desired outcome or purpose. Because MEMS devices are manufactured using batch fabrication techniques similar to those used for integrated circuits, unprecedented levels of functionality, reliability and sophistication can be placed on a small silicon chip at a relatively low cost [8], [9], [10].

Figure 4 illustrates how small these devices can be by comparing a gear set which is produced using MEMS to a mite [8].

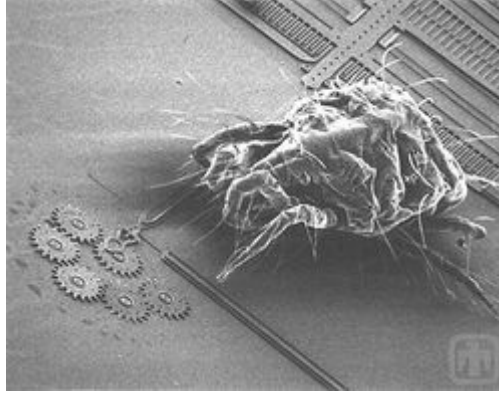


Figure 4. Gear set produced using MEMS compared with a mite (From Ref. 8).

D. SUMMARY

The multiple advantages of a wireless sensor network consisting of small and inexpensive motes versus a similar network consisting of micromotes was discussed at the beginning of this chapter, and it was explained why the wireless sensor network are more reliable and effective. Also the different existing applications that draw on this evolution, such as military, environmental, home intelligence, agricultural, and health monitoring have been briefly mentioned. Three different technologies that contributed to wireless sensor network development, digital circuitry, wireless communication and MEMS, were discussed and all together laid the necessary background to introduce the specific motes used in Chapter III.

III. SENSOR MOTE TYPES

This chapter will provide a brief introduction to the hardware used in this thesis as well as the layout and setup of a wireless sensor network. In this network, three Crossbow Micaz motes along with a base programmer and station which consisted of one Crossbow MIB510 and a Micaz attached to a desktop PC. The above components created the wireless sensor network from which traffic was captured and categorized based on the active message packet type into three major categories: routing, data and broadcast. The sniffing device used to capture the traffic was a Crossbow TelosB mote attached to a notebook computer. This arrangement provided mobility and allowed the packet sniffer's distance and position to be changed in relation to the WSN.

A. MOTE TYPES AND DESCRIPTION

1. Micaz mote

Figure 5 shows the basic components of a Micaz mote along with the circuitry layout. The mote consists of an Atmel ATmega128L microprocessor, a Chipcon CC2420 radio and antenna, external flash memory, a 64-bit Serial ID, a 51-pin connector, power connectors and indicator LEDs [1].

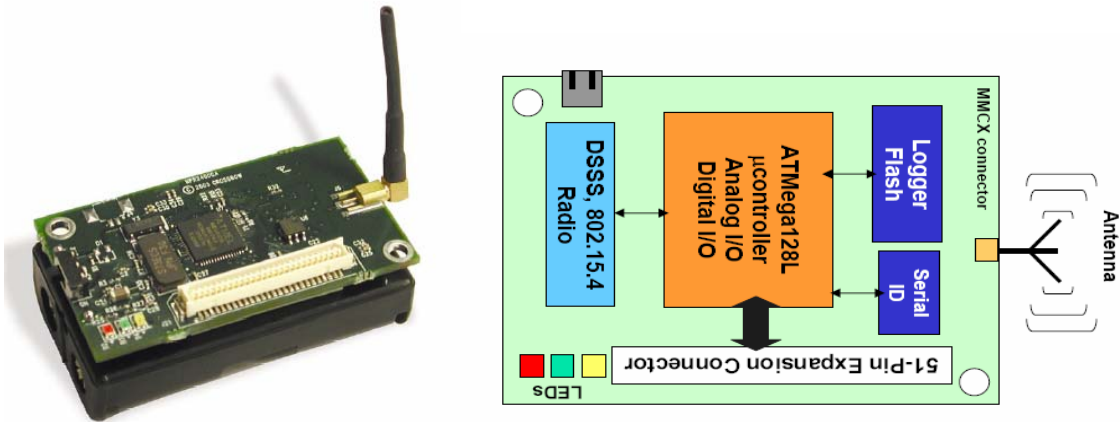


Figure 5. Micaz hardware description (From Ref. 1).

Table 2 provides some performance characteristics of the Micaz mote as mentioned in the paragraph above [1].

Micaz Platform	
Microprocessor: Amtel ATmega128L	Memory: 128Kb of flash 4Kb of SRAM
Radio: Cripcon CC2420	Radio: 250Kbps data rate Encoding: DSSS Modulation: O-QPSK Distance: Up to 135m, LOS, 1/2 wave dipole antenna Freq: 2400-2483Mhz
External Serial Flash Memory	512 Kb
51-pin expansion connector	Eight 10-bit analog I/O 21 general purpose digital I/O
Power options/Energy consumption	2 AA cells, through 51-pin or 2-pin Molex Energy usage: 1.8V x (5-10) μ A Energy sleep: 1.8V x 1 μ A

Table 2. Micaz performance characteristics. (From Ref. 1)

2. Telos Rev B Mote

Figure 6 illustrates the TelosB mote; where as described above, some of the differences from the Micaz mote are exhibited. The TelosB mote connects to a PC through a USB port/adaptor, has a different microprocessor (MSP430) and a 6-10 pin connector [1].

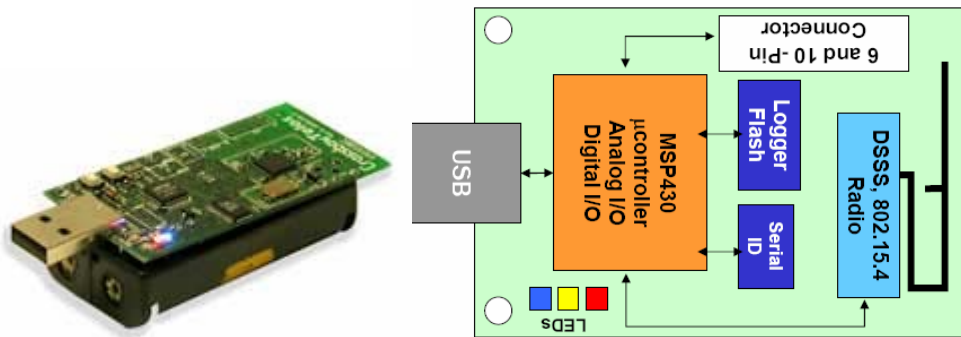


Figure 6. TelosB hardware description (From Ref. 1).

Table 3 provides the basic performance characteristics of the Telos rev B mote that has been used as a receiver for capturing the 802.15.4 traffic [1].

TelosB Platform	
Microprocessor: TI MSP430F1611	Memory: 48Kb of flash 10Kb of SRAM
Radio: Cripcon CC2420	Radio: 250Kbps data rate Encoding: DSSS Modulation: O-QPSK Distance: Up to 125m, LOS W/PCB inverted 'F' antenna Freq: 2400-2483Mhz
External Serial Flash Memory	1024 Kb
16-pin expansion connector	
Power options/Energy consumption	2 AA cells USB Energy usage: 1.8V x (5-10) μ A Energy sleep: 1.8V x 1 μ A

Table 3. TelosB performance characteristics. (From Ref. 1)

3. MIB 510 Programming Platform

In order to be able to program the motes with the desired application and to be able to make changes to their power settings, group ID's, mote ID, and any changes that are needed to be uploaded to the motes, a programmer board is needed.

In Figure 7, a block diagram of the MIB510 programming board is laid out. It has an RS-232 port, which is the programming communication link to a PC or any other external device that holds the application programs that are needed to be uploaded in order to operate the mote. The MIB 510 has an on-board in-system processor (ISP), an Altera 16L, which actually programs the mote as follows: the application code is downloaded to the ISP via the RS232 port, while the latter continually monitors/listens to the incoming serial packets. It should also be mentioned that the MIB 510 facilitates the power needs for the mote that it is attached to and TinyOS has to be installed on the PC in order to program the motes [1] (a detailed description of a TinyOS can be found in Chapter V).

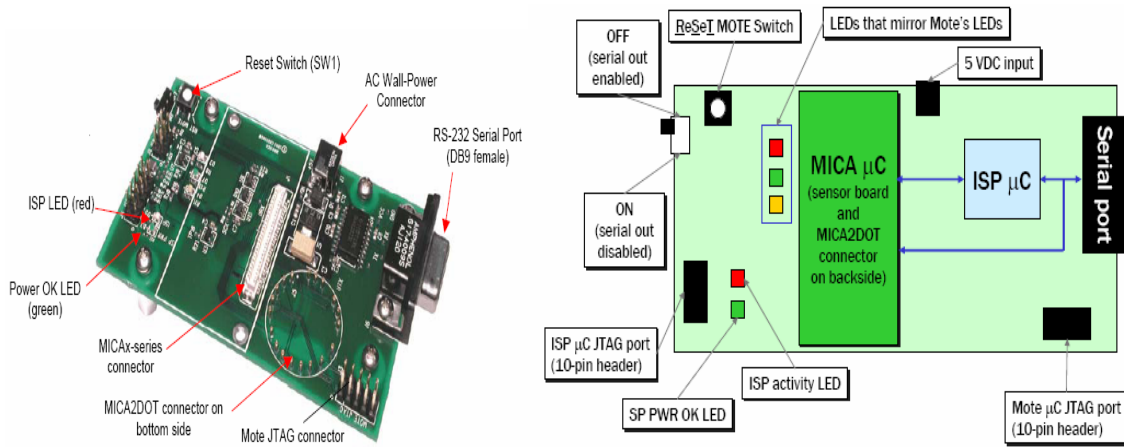


Figure 7. MIB510 Serial Interface Programming board (From Ref. 1).

4. MTS 310 Multipurpose Sensor Board

The sensor board that was used in this experiment was the MTS 310. As shown in Figure 8, this sensor board consists of a cluster of six sensors that are briefly described in the following paragraphs:

a. *Microphone and Tone Detector*

This can be used for acoustic ranging and acoustic recording and measurement. It consists of a pre-amplifier and a second stage amplification achieved with a digital pot control.

b. *Temperature Detector (Panasonic ERT-J1VR103J)*

A light detector based on a CdSe photocell with maximum sensitivity at the light wavelength of 690 nm.

c. *Two-axis Accelerometer (ADI ADXL202)*

A MEMS surfaced micro-machine that is used for tilt detection, movement and vibration. It has a g measurement range of $\pm 2g$ ($1g = 9.81m/sec$) and a resolution of 2 mg.

d. *Two-axis Magnetometer (HMC1002)*

A silicon sensor that consists of a very sensitive NiFe coating. It causes the bridge resistance to change when the magnetic magnitude is altered. It can detect moving objects like automobiles at a radius of 15 feet.

e. *Sounder*

This device is a 4 kHz piezoelectric resonator.

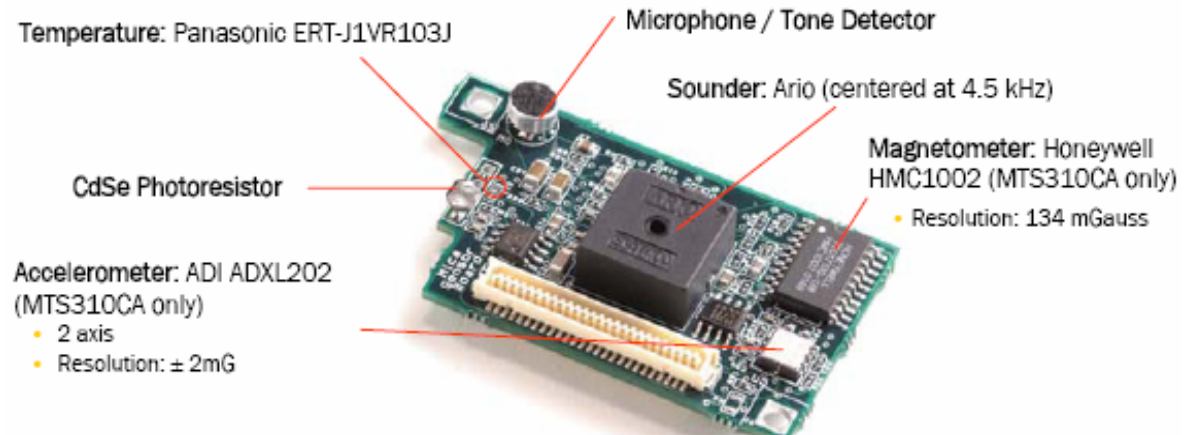


Figure 8. MTS 310 Multipurpose sensor board (From Ref. 1).

B. EXPERIMENT LAYOUT

In order to record the measurements for this experiment, the following deployment and devices were used, as mentioned above and illustrated in Figure 9.

As shown for the traffic capturing purposes a laptop with a TelosB directly attached was used, a configuration that provided mobility. The Wireless Sensor Network consisted of a base station – a MIB510 programming board with a Micaz mote attached on top base station and two additional Micaz's with MTS 310 sensor boards attached on top and powered by two AA batteries.

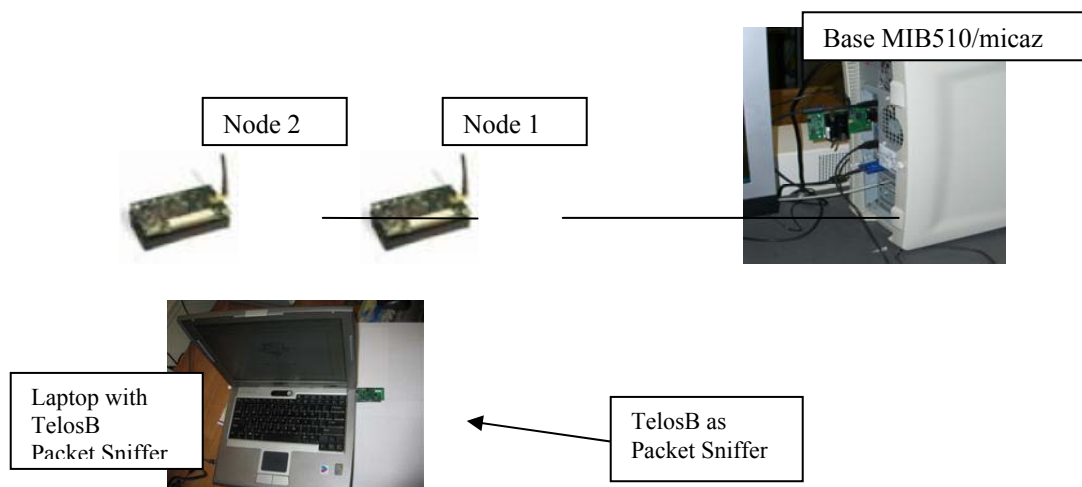


Figure 9. Experiment layout with a base station on the top right.

The two motes in Figure 9, where initially close to the base and had direct communication with it. Then they were purposely set at a greater distance from each

other and from the base in order to force them to create a multi-hop relationship of parent-child. This arrangement brought about the need for routing decisions to be made, and in consequence, the generation of routing traffic. This means that Node 2 was child to Node 1, and the latter transmitted the aggregated data to the base. The data was received at the base solely from Node 1. Finally, the relative position and distance of the laptop was changed, which held the packet capturing device, to test the effect of the movement of the elements above for the quality of the packet capturing.

C. SUMMARY

In this chapter, the two different configurations where traffic was intercepted were introduced, along with the criteria on how they were divided into two major categories; depending on the communication properties. Those were direct and parent-child scenarios. The first one identifies the communication scheme, where both nodes were directly exchanging data with the base station, while in the second scenario Node 1 relays all the packet exchange from Node 2 to the base. Continuing with the description of this sensor network, it was explained how the network is divided into two basic components: sensor nodes and the base station. The sensor nodes used were Micaz with a XMT310 sensor board, from Crossbow technologies Inc. Their basic specifications and capabilities were listed in Table 2 and Figure 5. Next, Table 3 listed the basic specifications for TelosB nodes, which were used as the hardware part for the packet sniffer.

The following chapter provides an in-depth analysis of the Physical (PHY) and Medium Access Control (MAC) packet formats and their content, and a brief description of TinyOS and the IEEE 802.15.4 protocol stacks.

IV. SENSOR NETWORK TRAFFIC TYPES

In this chapter, a brief description of the IEEE 802.15.4 standard, TinyOS operating system and some protocols is presented, which will facilitate the analysis of the different types of traffic produced in a WSN.

A. 802.15.4 AND ZIGBEE ARCHITECTURE

Figure 10 illustrates, according to the OSI model layer stack, the architecture of IEEE 802.15.4 and Zigbee Alliance. Starting from the bottom layer and moving up, we have the Physical Layer followed by the Medium Access Control Layer, which are standardized by the IEEE 802.15.4 Working Group. The Data Link, Network and Application layers are defined by the Zigbee Alliance. On the top of the layer stack is the Application Layer, which can be developed by different vendors, so it can support any application's needs [1].

802.15.4/ZigBee Architecture

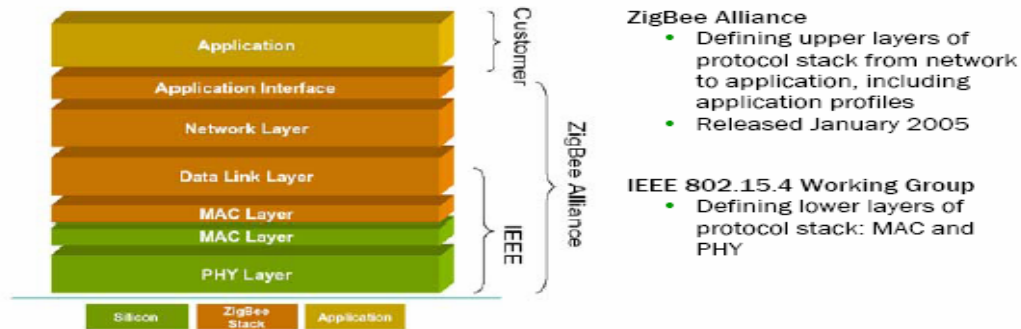


Figure 10. IEEE 802.15.4 and Zigbee Layer Architecture (From Ref. 1).

In this thesis, the layers that will be discussed are the Physical and Medium Access Control, since they contain all the information needed to categorize the traffic. Here it is important to mention, that those two layers are defined by IEEE 802.15.4, and a vendor or developer builds a package for the upper layers - Data Link to Application layer - which actually handle the manipulation of the two lower layers. If extra functionality and improvements are required to specific applications, they are facilitated with changes and modifications to the five higher layers.

The coexistence of the TinyOS and Xmesh sharing the same Physical and Medium Access Control layers is shown in Figure 11. Here on top of the PHY and MAC layers, two different layer stacks arise, as well as an interconnection bridge that facilitates the interconnection and communication between those two protocols suites [1].

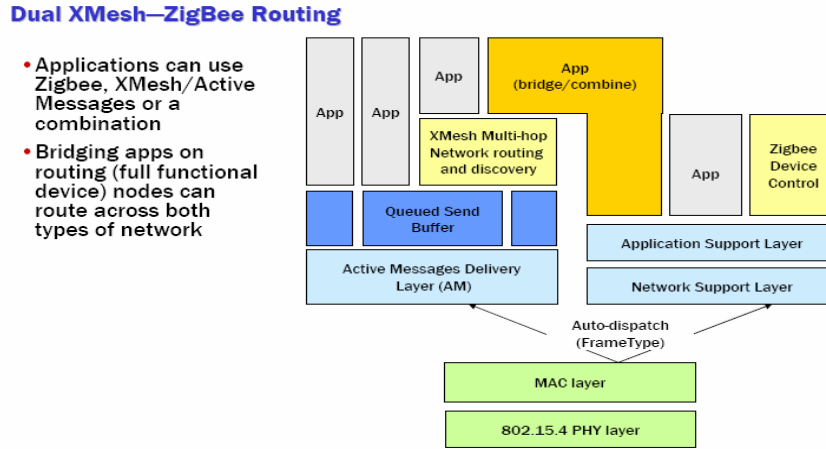


Figure 11. Xmesh and Zigbee protocol layering share the same MAC and PHY lower layer (From Ref. 1).

A description of the PHY and MAC layers within the Tiny MicroThreading Operating System (TinyOS) is addressed in the following paragraphs. It is shown that the Active Message type used by Xmesh in TinyOS is very similar to the TCP port. Different Active Message types imply different network services, and so different types of traffic.

The Crossbow Micaz and TelosB motes use the TinyOS research platform as a small operating system. From the Crossbow website [11]:

TinyOS is an *open-source operating system* designed for *wireless embedded sensor networks*. It features a *component-based architecture* which enables rapid innovation and implementation while minimizing code size as required by the severe memory constraints inherent in sensor networks. TinyOS's *component library* includes network protocols, distributed services, sensor drivers, and data acquisition tools – all of which can be used as-is or be further refined for a custom application. TinyOS's event-driven execution model enables fine-grained power management yet allows the scheduling flexibility made necessary by the unpredictable nature of wireless communication and physical world interfaces.

B. TINYOS/XMESH PACKET STRUCTURE AND ANALYSIS

The information exchange between nodes and the base station is achieved with transmission of many small-sized packets. Figure 12 illustrates the format of one packet and its different fields. The process that takes place before a packet transmission, in respect to the lower layer (physical layer) is as follows: there is a delay up to 15 packet lengths of time, where a packet time equals the time needed for a packet to be transmitted. Next a delay of 250ms is followed by a preamble, which helps the receiver to synchronize its timer to the incoming data. The preamble ends with the Sync (synchronization) portion of the packet that informs the receiver it should be receiving the packet. The actual packet data follows which is 36 bytes in length for the Mica2 (41 for the Micaz) [1].

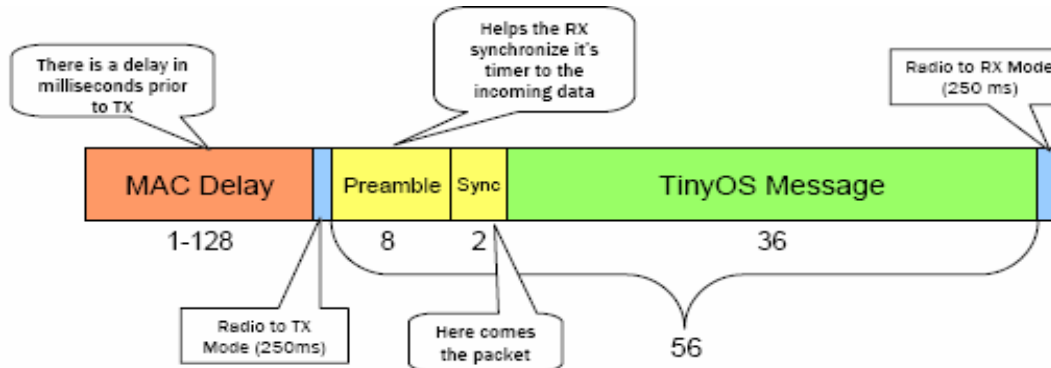


Figure 12. TinyOS Physical Message Packet (From Ref. 1).

The green portion of the packet in Figure 12 maps to the TinyOS packet, which is the Medium Access Control (MAC) layer packet before it is encapsulated into the physical layer. It is divided into three parts and has total length of 41 bytes (for the Micaz), but it can be extended if needed up to 125 bytes (125 bytes long is supported by IEEE 802.15.4) [1].

So starting with the first TinyOS section, the header has a length of 10 bytes and is further divided as follows. The first subdivision of the header section of the TinyOS packet structure is the Payload Length which has one byte of information and identifies the length of the payload carried in this packet. The Frame Control follows with two bytes and next is the Sequence Number which is one byte. The Sequence Number is used

to discard packets that have already been received and so protects from replay attacks. If a packet is captured and replayed afterwards the sequence number would have been already used and the packet is discarded. The Destination PAN ID has two bytes and it is the 16-bit address of the mote. This is manually programmed to each mote, and ranges from 01 to 65000 with node address 0 usually reserved for the base. The Origin Address occupies two bytes and it stands for the sender address. Next, the TOS AM Type (highlighted in Figure 13 in green) uses one byte, and as has been explained already, provides the information regarding to the type of the packet, and will be used as one of the criteria to categorize the traffic type. Finally, the last field in the header section is the TOS Group ID which is similar to the network address of a group of motes, assigned to a cluster of motes performing a specific task. This allows the use of the same radio channel by several different groups of motes. In the case where a mote receives a packet with Group ID that does not match its own, it discards the packet [1].

The next section of the TinyOS packet is the payload that can hold up to 29 bytes of user or application data and the last section is the Cyclic Redundancy Check (CRC) that has two bytes and is used to check for bit errors [1].

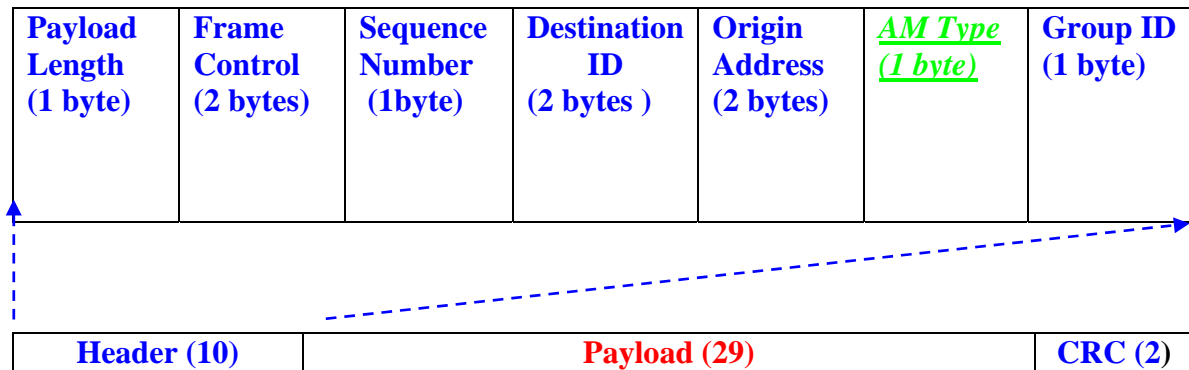


Figure 13. TinyOS Packet Structure (From Ref. 1).

Upon collection of the data to be transmitted, the TinyOS operating system running the Xmesh application takes the following message flow: Data or other information fills the message payload portion of the packet and then it is passed down the stack where the address of the next single hop destination is added. Continuing, the Active Message Type is specified. The complete packet with the format and additional

information as illustrated in Figure 13 is handed to the transmitting radio device, where it is delayed for up to 15 packet times (each packet time is about 4msec). Then a check to determine if the channel is clear is performed, where the channel access mechanism used is Carrier Sense Multiple Access Collision Avoidance (CSMA/CA). If the channel is free transmission occurs [1].

Now while in reception, the radio device upon the detection of the MAC header interrupts transmission, and reads the packet coming through the First In First Out (FIFO) buffer. Next verification that the sequence number received by the specific mote is correct occurs and if so the mote proceeds to read of the Group ID. If the group ID does not match, the packet is discarded. If the Group ID is correct the Active Message field is examined and depending on the AM type it is handed to the specific application [1].

A list with the Active Message Types and associated applications is shown in Table 4. In this experiment the AM numbers 3, 51, 246, and 250 were observed from the WSN while running Xmesh and the XMT310 applications. These AM numbers differ from application to application, and from vendor to vendor. For the Crossbow hardware and software components, as mentioned above, the AM 3 corresponds to Health packets from motes to the base, AM 51 corresponds to the environmental sensor measurements sensed from each mote (XMT 310 application), AM 250 is routing information packets, and AM 246 is an acknowledgement packet from base to the nodes [12].

Name of AM Type (#Define)	Value of AM_Type	Description
AM_HEALTH	3	Reserved for Health packets from the mote
AM_DATA2BASE	11	Upstream data msg from node to base, no end-end ack
AM_DATA2NODE	12	Downstream data to node
AM_DATAACK2BASE	13	Upstream guaranteed delivery to base
AM_DATAACK2NODE	14	Downstream guaranteed delivery to node
AM_ANY2ANY	15	Any to any
AM_MGMT	90	OTAP status message
AM_BULKXFER	91	OTAP transfer fragment
AM_MGMTRESP	92	OTAP status acknowledgement
AM_TIMESYNC	239	Time Sync packet
AM_PREAMBLE	240	Low power preamble packet
AM_FASTJOIN	241	Fast join
AM_FASTJOINRESP	242	Fast join
AM_DOWNSTREAM_ACK	246	Ack message from base to node
AM_UPSTREAM_ACK	247	Ack message from node to base
AM_PATH_LIGHT_DOWN	248	Light full power path down to node
AM_PATH_LIGHT_UP	249	Light full power path up to base
AM_MULTIHOPMSG	250	Route update message
MODE_ONE_HOP_BROADCAST	251	Single-hop service via XMesh
AM_HEARTBEAT	253	Base station heart beat message

Table 4. Identification Number for Active Message Types (From Ref. 12)

Once the motes are deployed and powered they initiate transmissions to the all broadcast address, trying to enter any network which they have not joined. Upon reception of the route update message from the mote, the base station also transmits a route update message, and informs the mote that the base station is there and is able to hear the mote. The same message then goes from the mote to the base. Upon the reception of the last message, the mote and base have established communication and exchanged all the other information needed to start passing data. In the case that the base

is out of the radio coverage area of the mote, the same kinds of messages are exchanged between the mote and another mote which might be within radio distance [11]. In that case each mote listens to its neighbor and uses this information to populate a routing table of up to 16 neighbors (except for the base which builds a routing table of 40 neighbors). This relationship, which serves the relay of traffic between motes to the base, is called the parent-child relation. This decision considers which neighbor requires less energy to send the traffic to, and the minimum number of hops to the base. The maximum number of children assigned to a mote is 50 and it can go up to 100 for the base. Typically the interval of the Route Update Message, which as mentioned above has AM 250, is 360 seconds for low power consuming transmission, and 36 seconds for high power. If no Route Update Message is received within a defined interval the mote will reboot [10].

The Route Update Messages (RUM) contains 4 basic information fields. The first information field contains the Parent ID number, or if the node has not joined the mesh network yet a 0xffff (broadcast) is transmitted. The second field is used for the cost that is needed to send a message to the base, followed by the number of hops necessary to reach the base. Finally in the last field is a list of the five most qualified neighbors as mentioned in the paragraphs above. Since each node keeps a routing table of 16 neighbors, 4 RUM messages of sequential counting are required to be sent [12].

The next type of packet observed in this experiment is the Health packet. There are two types of Health packets, the Statistics Health packet and the Neighbor Health packet. These are transmitted in an alternative fashion on every health update interval and have the same packet format [12]. The Active Message number ID for these is AM 3.

The health packets include statistical information about the packets being sent by this mote, like number of packets transmitted, number of dropped packets and number of retransmissions. They also contain power information such as voltage readings and accumulated power usage. Finally, they include information about parent node ID, link quality and path cost. Neighbor health packets contain information about the mote's neighbors and the link quality to these nodes, which is estimated by monitoring the number multihop headers and processing this with an exponentially-weighted moving average algorithm [12].

There are two kinds of packet acknowledgements, the link level acknowledgement and the end-to-end. Data packets use the link level acknowledgement which requires retransmission between a sender and receiver until an acknowledgement is being received by the sender. Systems and application that require energy efficiency but not 100% delivery of data should use it. End-to-end acknowledgement packets are used by the health packets. In this case the base must receive and acknowledge the originator mote [12].

Table 5 summarizes the properties listed above for different kinds of messages as well as the differences between the base station and the rest of the motes.

Parameter	Default Value	Description Summary
Data length	29 Bytes	Maximum data for a TOS packet. It can be change with appropriate applications.
Base Station Address	0	Node ID (or address) for base station. Should never been changed
Route Update Interval	60 sec for High Power 600 sec for Low Power	Time Between health messages
Descendant Table Size	Base : 100 Motes : 50	Defines the maximum number of children to a mote.
Data Message Rate	10 sec High Power 180 sec Low Power	Defines the transmission time interval for the data messages.
Route Table Size	Base : 30 Motes : 15	Defines maximum Number of neighbors that a mote can track. A mote that is not listed here can not be selected as a Parent node.
Queue Size	Base : 16 Motes : 8	Define maximum number of messages that can be queued. Once exceeded packets will be dropped.

Table 5. Summary of the Base and Motes messages properties and differences

C. SUMMARY

In this chapter the MAC packet is parsed down into the elements that are used, in Chapter VI for traffic categorization. Through this process, it was illustrated that the MAC packet for the Micaz motes is 41 bytes in length, with a header of 10 bytes. The latter field included the following information: destination and originating addresses, group ID, active message index, and application information. Next, it was explained how different types of active message indices can identify different kinds of packet types, like routing, data, health and acknowledgement. Table 4 illustrated some of the possible active message types that can be found in the applications of the motes used in this thesis. Finally the properties and format of the routing, data, health and acknowledgement

packets are explained, since those were seen in this thesis. An example on how each of these is seen from the output of the packet sniffer follows in the next chapter as well as how the initial packet sniffer program was altered to serve these needs.

THIS PAGE INTENTIONALLY LEFT BLANK

V. TRAFFIC CAPTURE SNIFFER DESIGN

As introduced and analyzed in Chapter IV, the scope of this thesis is to categorize the traffic, mainly according to the AM number ID contained in each packet exchanged within the WSN. Other data, such as payload data length, number of packets, interarrival time, and different AM type and node type, are also going to be used to extract statistical information and build a database with behavior patterns for a specific location, environment and number of nodes. In order to facilitate this need, an existing packet sniffer [4] was used which was written initially by Hong-Siang Teo and was modified to meet the needs of this thesis.

This sniffer was modified with the addition of several java programs (classes) named `activeMessage.java`, `displayWindow.java` and `outputtoCsv.java` respectively as well several code lines in the *Sniffer* class. Descriptions of these three classes as well as their functionality will be presented in the following paragraphs in this chapter, and the source code is included in Appendix A.

A. SNIFFER ANATOMY

The packet sniffer consists of hardware and software components. The hardware components were discussed in Chapter III, so no further description is needed. In Figure 14 a block diagram of the packet sniffer is illustrated. From the left to right, we can see the WSN, which for this experiment consisted of three Micaz's, whose traffic was captured using a TelosB mote as the packet sniffer hardware. The TelosB has been "flashed" (programmed) with `TransparentBase`. `TransparentBase` is a TinyOS application located under `apps` directory within `Tinyos-1.x` directory. The complete path is `tinyos/cygwin/opt/tinyos-1.x/apps/TransparentBase`. This application captures all the packets that it can hear and reports them back to the UART. It also forwards all incoming UART messages out to the radio [13]. It is also a bidirectional bridge between the radio of the TelosB and the serial port, which has the ability to ignore the group ID and AM type and forward all packets to the PC (so all packets from all WSN in range will be captured with no limitation to tune the sniffer to a known group ID).

Within the PC the Serial Forwarder application instantiates a server which provides a bi-directional packet stream between a mote connected to the host PC and clients anywhere on the network [14]. SerialForwarder is located in *tinyos/cygwin/opt/tinyos-1.x/tools/java/net/tinyos/sf* directory, and it basically allows the communication of the application through a network interface rather than a serial interface. Some operating instructions and settings, which are required for the normal operation for the Serial Forwarder, are described in Appendix B.

The last component of the packet sniffer is the java application *Sniffer* which extracts the information delivered by SerialForwarder, such as protocol and application information, and outputs this in a form which enables and allows further processing and analysis [4]. This is where this thesis has added three different java applications in order to facilitate the traffic categorization.

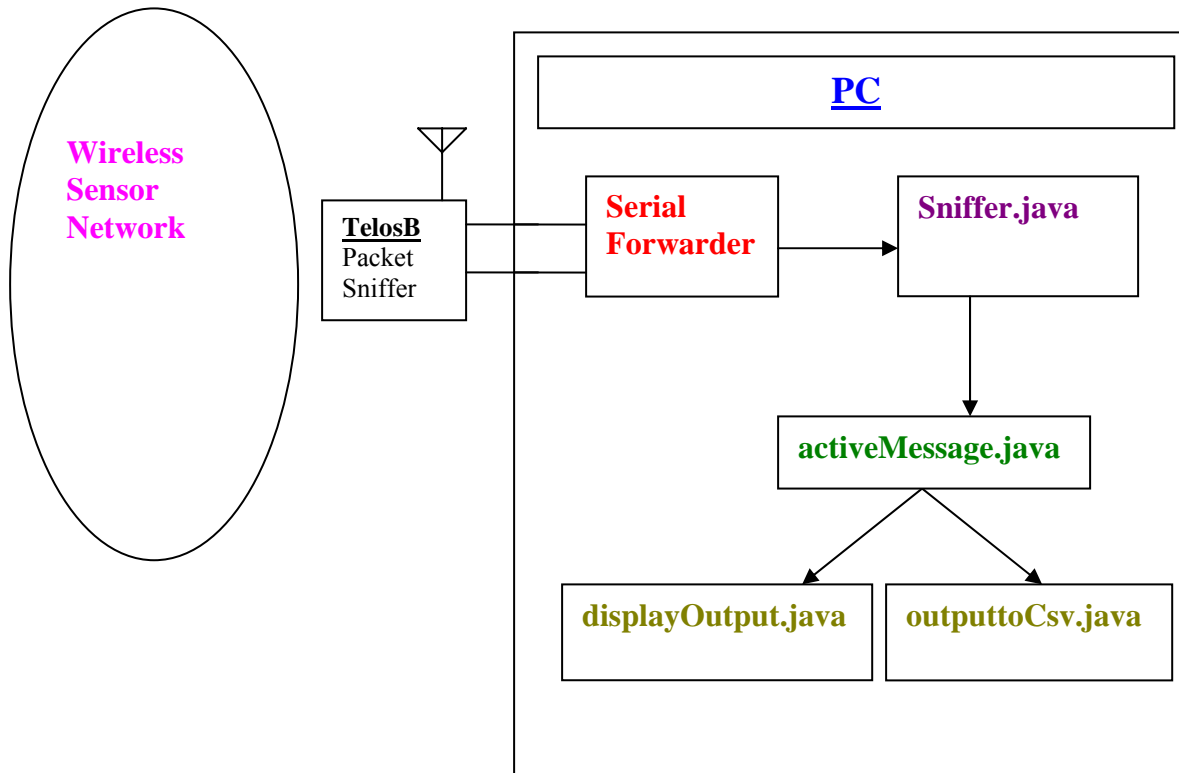


Figure 14. Sniffer application block diagram.

B. DESCRIPTION OF JAVA CLASSES

The `activeMessage.java` is a program that uses the java tool `java.util.StringTokenizer` and allows the programmer to parse the elements of a larger string, such as the incoming TinyOS MAC packet, into as small elements as are needed. Incoming packets had the following structure: 17:07:46.781 TOS 125.65535.51 (27) [010001001200ff8481ffffae01cd014f03a001010001001c031c03]. Comparing this to the TinyOS MAC packet format and with the use of reverse engineering, an identification of each part was achieved.

The packet output started with the time stamp, which in this example was 17:07:46.781, which was assigned the string name `packetTime` by the `activeMessage` program. The next part is the string “TOS,” which was used to display the protocol information (TOS is an abbreviation for TinyOS) and was not needed for this experiment. This was followed by 125.65535.51. Those numbers were parsed down to three additional parts as follows: “125” corresponds to the Group ID and was given the string name `packetGroupID`, “65535” was the destination address (in this case broadcast) with the assigned string name `packetDestination` and finally “51” was the active message number with string name `packetAM`. Next followed “(27)” where the number within the parentheses corresponded to the payload length. It was assigned to the string name `numberOfBytes`. The packet ended with the data portion where only the originating address was determined which corresponded to the first 5 digits within the bracketed string. In this example “[01000” was assigned to the string name `PacketNoteID`. Note here that comparing the received information to the packet format we expected to see, the last part, which is the Cyclic Redundancy Check (CRC), is missing. The later was not needed for the traffic profiling, so no further investigation of this part was conducted.

The output of the `activeMessage.java` program was then given to the `outputtoCsv` and `displayWindow` programs for further processing. The `outputtoCsv` exports the above output to a CSV file, with values separated by commas, so it can be imported to an Excel spreadsheet for further analysis. There are two settings that the user might be required to change depending on user preferences. These are where this file should be stored and how many lines, which is equivalent to the number of packets, should be saved in each

CSV file. Starting from the latest, Excel spreadsheets have a limitation of 65535 lines so this program is configured to open a new CSV file every 60000 packets (or lines).

The following code, which also listed in Appendix A paragraph B, points to the fields that the user can change depending of the number of packets the user might want to input into a file as well location that the files should be stored.

```
String Filename="complete path where user wants to save the files";
    public void printTraffic (String packetGroupID, String
packetDestination,String packetAM,String packetAM,String packetNodeID, String
numberOfBytes )
    {
        count ++;
        if (count == value of number of lines before opening new file)
        {
            Filename+="1";
        }
        File cvs = new File (Filename);
```

A good common practice, considering the purpose of this program, will be to set this value to 1000 packets or less, in order to run an analysis of the input traffic in short time intervals and determine any discrepancies that might have occurred. This topic will be addressed in more detail in Chapters VI and VII.

The outputtoCsv java class opens a GUI window to enable the user to have a real time display of the incoming packets. In this program layout, where the WSN consisted of only few motes, the interarrival time of the packets was about 0.19 seconds and the user was able to observe any anomalies in the traffic. This is not realistic because in reality the WSN will consist of a minimum of hundreds of motes and the interarrival time will drop tremendously.

C. OUTPUT

Figures 15 and 16 illustrate the displayWindow Graphical User Interface (GUI) and the CSV file format, respectively. Starting from Figure 15 we can see that this window is labeled “Traffic Categorization” and displays six columns which are respectively: GroupID, Destination (address), AM (active message number), Time, Node ID (originating Node ID number), and finally Number of Bytes.

Group ID	Destination	AM	Time	Node ID	Number of Bytes
125	0	3	13:39:16.000	02	12
125	0	51	13:39:16.296	01	27
125	0	51	13:39:17.171	02	27
125	0	51	13:39:18.562	01	27
125	0	51	13:39:19.203	02	27
125	0	3	13:39:19.625	01	12
125	1	246	13:39:19.640	00	11
125	0	51	13:39:20.609	01	27
125	0	51	13:39:21.234	02	27
125	0	51	13:39:22.625	01	27
125	0	51	13:39:23.281	02	27
125	0	51	13:39:24.921	01	27
125	0	51	13:39:25.531	02	27
125	0	51	13:39:27.125	01	27
125	0	51	13:39:27.796	02	27
125	0	3	13:39:27.828	02	24
125	2	246	13:39:27.843	00	11
125	65535	250	13:39:28.062	00	21
125	65535	250	13:39:28.687	02	18
125	0	51	13:39:29.390	01	27
125	0	51	13:39:29.843	02	27
125	0	51	13:39:31.421	01	27
125	0	3	13:39:31.671	01	24
125	0	51	13:39:31.875	02	27
125	65535	250	13:39:33.343	01	18
125	0	51	13:39:33.453	01	27
125	0	51	13:39:33.906	02	27
125	0	51	13:39:35.484	01	27
125	0	51	13:39:36.218	02	27
125	0	51	13:39:37.750	01	27
125	0	51	13:39:38.531	02	27
125	0	3	13:39:39.640	02	12
125	2	246	13:39:39.640	00	11
125	0	51	13:39:39.906	01	27
125	0	51	13:39:40.887	02	27
125	0	51	13:39:42.187	01	27
125	0	51	13:39:42.968	02	27
125	0	3	13:39:43.421	01	12
125	1	246	13:39:43.437	00	11
125	0	51	13:39:44.218	01	27
125	0	51	13:39:45.000	02	27
125	0	51	13:39:46.250	01	27
125	0	51	13:39:47.046	02	27

Figure 15. Illustration of the displayWindow java program; a GUI for traffic monitoring

The CSV file format is illustrated in Figure 16 and has the same elements as in the displayWindow GUI.

```

File Edit Format View Help
125 , 0 , 51 , 09:24:48.531 , 01 , 27
125 , 0 , 51 , 09:24:48.609 , 02 , 27
125 , 0 , 3 , 09:24:49.921 , 02 , 24
125 , 2 , 246 , 09:24:49.921 , 00 , 11
125 , 0 , 51 , 09:24:50.828 , 01 , 27
125 , 0 , 51 , 09:24:50.953 , 02 , 27
125 , 0 , 3 , 09:24:51.203 , 01 , 24
125 , 0 , 51 , 09:24:52.843 , 01 , 27
125 , 0 , 51 , 09:24:53.078 , 02 , 27
125 , 0 , 51 , 09:24:55.171 , 01 , 27
125 , 0 , 51 , 09:24:55.359 , 02 , 27
125 , 0 , 51 , 09:24:57.187 , 01 , 27
125 , 0 , 51 , 09:24:57.406 , 02 , 27
125 , 0 , 51 , 09:24:59.218 , 01 , 27
125 , 0 , 51 , 09:24:59.390 , 02 , 27
125 , 0 , 51 , 09:25:01.437 , 02 , 27
125 , 0 , 51 , 09:25:01.562 , 01 , 27
125 , 0 , 3 , 09:25:01.578 , 02 , 12
125 , 0 , 51 , 09:25:03.468 , 02 , 27
125 , 0 , 3 , 09:25:03.578 , 01 , 12
125 , 1 , 246 , 09:25:03.578 , 00 , 11
125 , 0 , 51 , 09:25:03.687 , 01 , 27
125 , 65535 , 250 , 09:25:04.906 , 00 , 21
125 , 0 , 51 , 09:25:05.750 , 02 , 27
125 , 0 , 51 , 09:25:06.031 , 01 , 27
125 , 0 , 51 , 09:25:07.750 , 02 , 27
125 , 0 , 51 , 09:25:08.046 , 01 , 27

```

Figure 16. Illustration of the CSV file format

D. SUMMARY

A brief description of Hong-Siang Teo's packet sniffer for TinyOS [17] compatible motes was discussed followed by a detailed description of the functionality and properties of the three java programs that were added by this thesis. It was explained that the *activeMessage.java* program is used to parse the incoming packets, and then is passed to *displayWindow.java* and *outputtoCsv.java* programs, where the first one served the real time visualization of the MAC header information, and the latter the output in a suitable format of a CSV file. As explained, the CSV format is easy to be exported to an Excel spreadsheet, which is very convenient for further analysis.

In the following chapter, it is shown how this packet information was grouped and categorized according to the Active Message type into four different categories, and what traffic profiles and patterns were discovered. Those patterns qualify a normal traffic behavior, which can be used to identify variations and anomalies when such network is under attack.

VI. TRAFFIC CATEGORIZATION FRAMEWORK

In Chapters IV and V, the hardware and software components of a packet capturer for a sensor network consisting of Crossbow Micaz motes was fully explained and analyzed. This packet capturer can be considered and used as the sensor component of a possible Intrusion Detection System (IDS), for a TinyOS network. In this chapter, the traffic of three motes in two different topologies, as described in Chapter III, will be captured, stored and analyzed in detail to provide a baseline for normal network behavior.

There are several different kinds of Intrusion Detection Systems (IDS), depending on the network and the location of the sensor. These are summarized as follows: network intrusion detection systems, protocol-based IDS, application protocol-based IDS, host-based IDS and finally hybrid IDS which is actually a combination or two or more of the above approaches [15].

This thesis examines a network IDS. There are two kinds of network IDS depending on the detection method – signature-based and anomaly-based. The signature-based IDS compares the network traffic to known attack patterns, and is supposed to detect all known attacks. An anomaly-based IDS, like the one developed in this thesis, scans the network traffic and analyzes it according to baselines and patterns of normal traffic behavior. The major advantage of this approach is that it can detect attacks that are not yet known, but it is subject to false alarms when the baselines are not accurate or when the administrator has assigned too narrow a deviation. Usually an IDS is composed of the following four parts: event generators (E-boxes), event analyzers (A-boxes), event databases (D-boxes), and finally response units (R-boxes) [15].

An event generator is the device that captures the traffic, incoming and outgoing, and then passes it to the other IDS components. The event analyzers process and analyze the data that is coming from the event generator, and create traffic profiles and traffic behaviors. The event databases simply store the data from the two previously mentioned components, so the system administrators can access it anytime. The last component is

the response unit, which provides countermeasures when network intrusion is detected. For this thesis, the event generator component is the TelosB mote programmed with the *TransparentBase* application.

Following the IDS component model introduced in the paragraphs above, the event analyzer follows and in this thesis can be mapped to the laptop on which the TelosB was attached. The applications that were necessary to perform the analysis and the statistical profiling of the incoming data are *Sniffer* and TinyOS. The incoming data was forwarded from the TelosB to *Sniffer* through *Serial Forwarder*. The *displayWindow.java* program provided a graphical User Interface for monitoring the traffic, and parallel the needed packet information were written into a CSV file with *outputtoCsv.java*. The statistical analysis was then performed offline, by importing the CSV to an Excel spreadsheet. All data was saved on the hard drive of the laptop which can be considered the event database, and finally no response unit component was developed.

In the following paragraphs, the statistics of two cases are illustrated as well as basic behavior patterns. Section A refers to the layout where two Micaz motes are directly communicating to the base node, while in Section B Node1 relays all traffic from Node 2 to base, as well as its own traffic.

A. TRAFFIC STATISTICS

1. Direct Motes to Base Connection

In this first part, a total of 15,703 packets were captured in a time period of three hours and five minutes. The distance between the motes and the base was four meters and between motes two meters. The whole experiment took place within a room, and the motes were not moved.

In Figure 17, the percentage of the traffic according to the active message type is illustrated. Of the 15,703 packets, 13.07% was with AM 3 corresponding to health type packets. Next, 70.97% of the traffic contained data packets, with AM 51. The last two types observed were acknowledgement packets with an AM 246, which were taking 10.09% of the total traffic, while the route update packets with AM 250 were found to be 5.85% of the traffic.

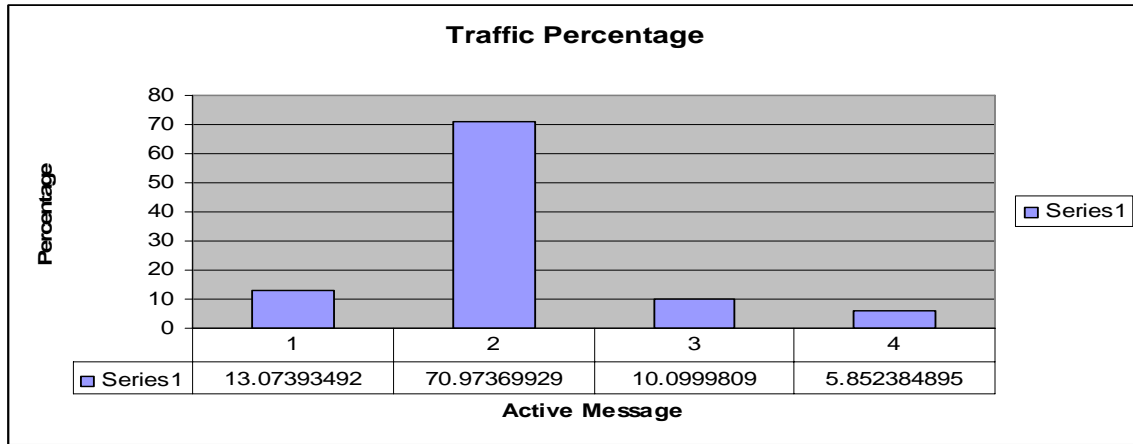


Figure 17. Traffic percentage statistics in accordance to Active Message type.

Continuing with the traffic profiling, the packets were grouped according to the Active Message index, for further analysis. Starting with the Health packets (AM 3), it was observed that all packets were initiated from Nodes 1 and 2, with destination Node 0 (base station) and had payload length of 27 bytes. More specifically the payload length started with 11 bytes and within the first packets stabilized to 27 bytes. Also the interarrival time at the base station for each node was 11.734 seconds for the messages originating from Node 1, and 11.735 seconds originating from Node 2.

In the data packets (AM 51) for this specific application and vendor, the same originating source as well as payload length was observed as in the Health packets. The interarrival time for these messages was 2.046 seconds for traffic moving from Node 1 to base and 2.047 seconds when coming from Node 2.

The acknowledgement packets (AM 246), were transmitted only from the base and had destination addresses of either one the two nodes or the broadcast (broadcast address is: 65535). The payload length for those packets was 11 bytes and the interarrival time interval from the base station to Node 1 was 12.219 seconds and to Node 2, 12.438 seconds.

The last packet type examined was the routing update packet (AM 250). Those packets were broadcasted by both motes and the base to a destination address 65535. The payload length was different for the base and nodes, with the base having payload length

21 bytes and the nodes 18 bytes respectively. The transmission interval time for the base station was 33.248 seconds, for Node 1 46.969 seconds and finally for Node 2 41.082 seconds.

Summarizing, as can be seen in Table 6, it was observed for this wireless sensor network setup that, depending on the packet type, specific constant values were used from both motes and the base, according to originating and destination addresses, and payload length. Any variation of those imposes integrity violations.

AM Type	Originating Node	Destination Node	Payload Length	Interarrival Time
AM 3 (Health Packet)	1, 2	0	27 bytes	11.735 sec
AM 51 (Data Packet)	1, 2	0	27 bytes	2.046 sec
AM 246 (Acknowledgement packet)	0	1,2, 65535	11 bytes	1 : 12.219 sec, 2 : 12.438 sec
AM 250 (Route update Packet)	0, 1, 2	Broadcast (65535)	0 : 21 bytes, 1, 2 :18 bytes	0 : 33.248 sec, 1 : 46.969 sec, 2 : 41.082 sec

Table 6. Traffic characteristics for all Active Message type on direct scenario.

Integrity violations can easily then imply confidentiality violation, because if someone is able to modify the packets content they likely can see and read the traffic. In addition several other situations might arise if malicious code is placed in the payload. The latter could execute a program in both the motes and the base causing the network to collapse or transmit false readings as well as to cause availability issues by forcing the motes to transmit in shorter intervals causing battery consumption.

2. Parent-Child Network Deployment

In this node deployment, where Node1 was relaying all traffic from Node 2 to the base the following statistics were observed. A total of 5552 packets were captured within an hour and as will be analyzed in the following paragraphs very similar statistics as above were observed.

Figure 18 illustrates the percentage of the packets captured according to the Active Message type. We can see that the Health packets were 12.55% of the total traffic, and the Data packets 72.94%. Continuing, the Acknowledgement packets occupied 10.82%, and finally the Routing update packets 3.67%.

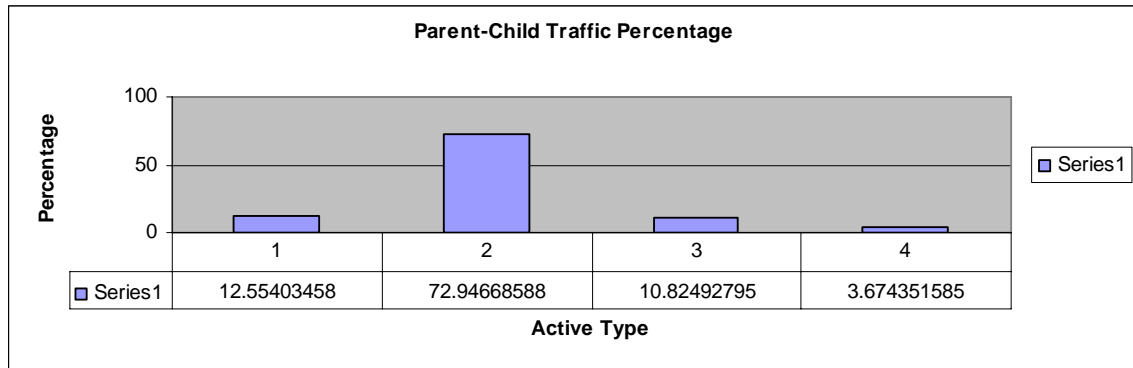


Figure 18. Traffic percentage statistics in accordance to Active Message type for the Parent-Child setup.

A comparison of these two setup variations is illustrated in Figure 19 where we observe a very small variation between the Health, Data and Acknowledgement packets. The routing packets show a large variation, from 3.67% to 5.85%, a variation equal to 62.78383%.

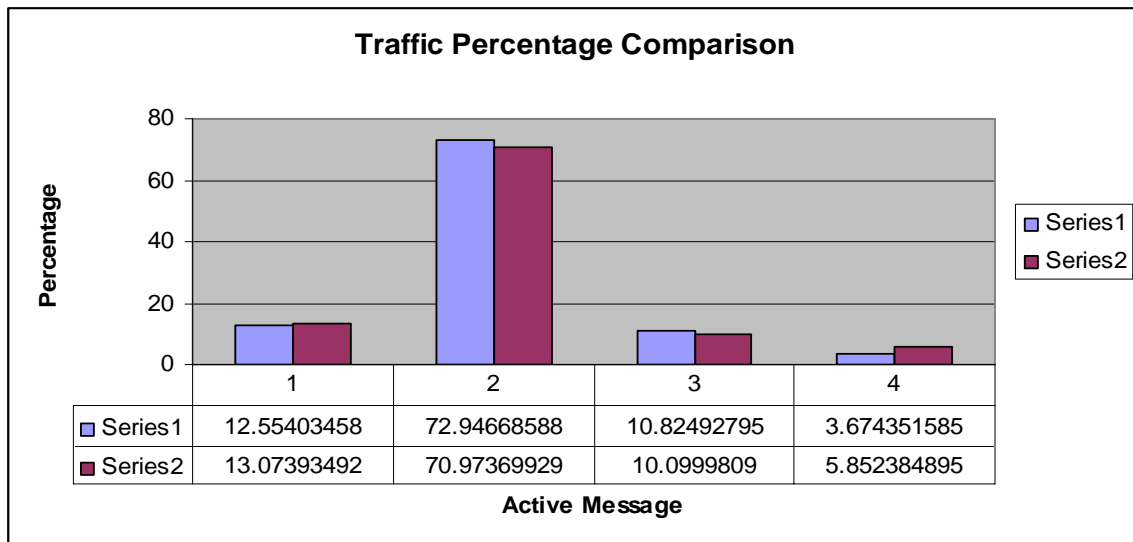


Figure 19. Traffic percentage statistics comparison in accordance to Active Message type for Direct Base-Motes communication and Parent-Child setup.

At this point, it is important to understand that different network deployments, number of nodes, environmental conditions affect some of the traffic characteristics and should always be considered by the network administrator.

Continuing, the analysis in this topology observed that the health packets (AM 3) were originated from Node 2 to Node 1 and then from Node 1 to the base station. Of course, in a larger network one would expect to see changes of the relay nodes but not to the nodes that relay to the base (since they are the ones in the closest vicinity to the base). Those packets had payload lengths of 12 and 24 bytes, and the interarrival time for the packets originating from Node 1 was 7.410 seconds, and for those originating from Node 2 18.655 seconds.

Next, the analysis of data packets (AM 51) found the same statistics as seen in the first scenario examined above, except from the originating and destination addresses, which was normal and expected. A small variation in the transmission interval time was observed, where Node 1's mean transmission time was 1.329 seconds, and for Node 2 2.861 seconds. This was expected due to the relay radio communication.

The acknowledgement packets (AM 246) were originated from the base station with destination Node 1 and broadcast, and from Node 1 to Node 2. That introduced an expected variation from the first scenario, where we had observed that only the base station was transmitting Acknowledgement packets. All these type of packets had payload length of 11 bytes. The transmission interval time from the base station to Node 1 was 10.325 seconds and from Node 2 to Node 1 was 12.218 seconds.

Finally, the routing update packets (AM 250) were broadcasted from all nodes with payload lengths 15 bytes for Nodes 1 and 2, and 18 bytes for the base station. The mean transmission interval time, when originated from base station was found to be 16.678 seconds, and for Nodes 1 and 2 was calculated to be 19.266 and 18.549 seconds, respectively.

Table 7 summarizes the traffic characteristics that were analyzed in the above on the parent-child radio communications scenario.

AM Type	Origin Node	Destination Node	Payload Length	Interarrival Time
AM 3 (Health Packet)	1→ 2→	→ 0 → 1	12 bytes 24 bytes	7.410 sec 18.655 sec
AM 51 (Data Packet)	1→ 2→	→ 0 → 1	27 bytes	1.329 sec 2.861 sec
AM 246 (Acknowledge- ment packet)	0→ 1→	→ 1, 65535 → 2	11 bytes	1 : 10.325 sec, 2 : 12.218 sec
AM 250 (Route update Packet)	0, 1, 2	Broadcast (65535)	0 : 18 bytes, 1, 2 : 15 bytes	0 : 16.678 sec, 1 : 19.266 sec, 2 : 18.549 sec

Table 7. Traffic characteristics according to active message types for the parent-child radio communication scheme.

In Figures 20 and 21, we can see the packet payload lengths for both the direct and parent-child deployments respectively, and the number of packets associated with each. Starting with Figure 20, the number of packets associated with each payload length is 1605 packets with payload length 11 bytes, 1045 packets with payload of 12 bytes, 534 packets with payload of 18 bytes, 358 packets with payload of 21 bytes, 1006 packets with payload of 24 bytes, and 11144 packets with payload of 27 bytes.

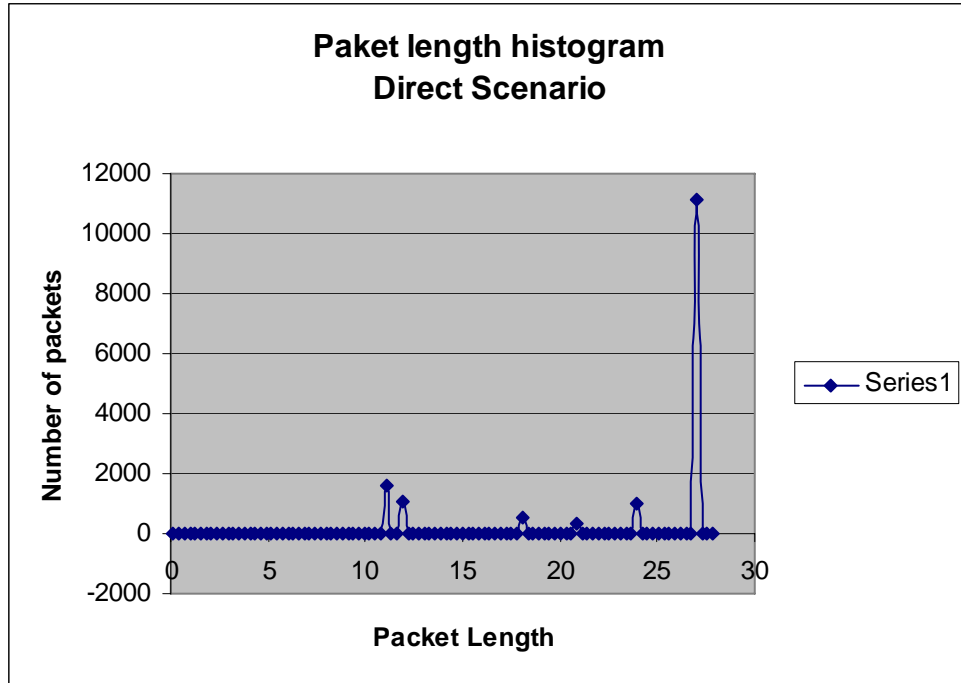


Figure 20. Number of packets vs. payload length in direct communication scenario.

In Figure 21, the number of packets associated with each payload length is, 601 packets with payload length 11 bytes, 310 packets with payload of 12 bytes, 93 packets with payload of 15 bytes, 108 packets with payload of 18 bytes, 387 packets with payload of 24 bytes, and 4050 packets with payload of 27 bytes. We should mention here that the packets with payload of 15 bytes exist only in the parent-child topology and that generally both traffic traces exhibit the same or similar characteristics.

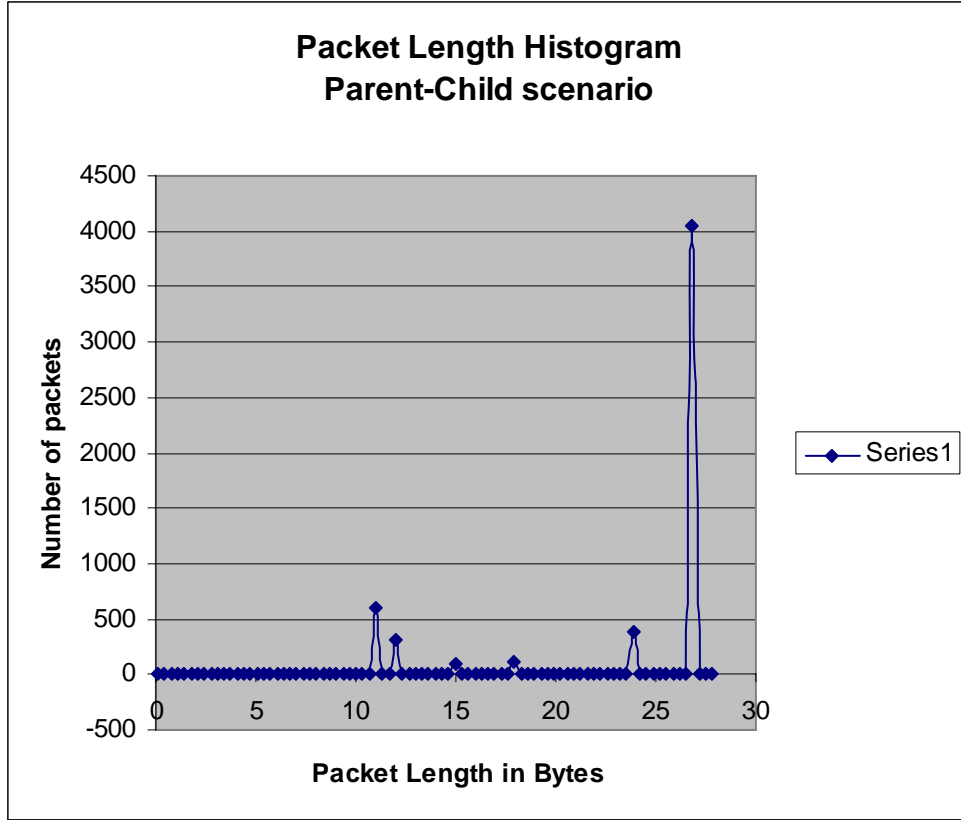


Figure 21. Number of packets vs. Payload Length in Parent-Child communication scenario.

In the above paragraphs, it was shown that when using the payload length as one of parameters of normal behavior, we expected to see packets of 11, 12, 15, 18, 21, 24, 27 bytes long. Other statistics as a percentage of traffic relative to Active Message index was proven to a reliable parameter. The above two parameters and how they are related to the associated addresses, originating and destination, node IDs, interarrival and transmission interval times, constitute unique traffic profiles.

B. ESTIMATING SELF SIMILARITY IN WSN

Another significant parameter for traffic analysis is the estimation of self similarity. Form ref [14]:

A phenomenon that is self-similar looks the same or behaves the same when viewed at different degrees of “magnification” or different scales on a dimension.

The parameter used to estimate self-similarity is the Hurst parameter (H), and provides a measure of its extent [20]. It was named after H.E. Hurst who studied water

levels of the Nile and other rivers. His research suggested that values of the Hurst parameter $H = 0.5$ indicate the absence of long-range dependence while for values of $H = 1$ indicate infinite long-term dependence [16]. In other words in the traffic examined above, for large values of H the longer it has been since the arrival of a data packet the longer it would be for the arrival of the next similar one, or if we want to express it differently the sooner a packet type of one kind has arrived the more likely it is for another one to arrive. For values of H less than 0.5 the arrival of a packet is independent of the arrival of the last packet of the same type.

Another parameter used to model the extent of long term dependence introduced in the paragraph above is the β parameter [16]. The long-range dependence is defined in terms of the autocovariance and it is rapidly decreasing with time. A short-range dependent process satisfies the condition that its autocovariance decays at least as fast as exponentially while long-range dependent processes have a hyperbolically decaying autocovariance [16]: $C(k) \propto |k|^{-\beta}$ as $|k| \rightarrow \infty$, $0 < \beta < 1$. The Hurst parameter is related to the β parameter with the following relation: $H = 1 - (\beta/2)$ [16]. As mentioned above self similar traffic has a Hurst parameter equal to: $0.5 < H < 1$ which means that the β parameter takes the following values: $0 < \beta < 1$.

There are three common ways to measure self similarity: Variance-Time plot, R/S plot, and the computation of Whittle's estimator from the spectral density [16]. For this thesis the first method was used.

A discrete random process x is said to be exactly self-similar with parameter β and $0 < \beta < 1$ if for all $m = 1, 2, \dots$ we have:

$$\text{var}(x^{(m)}) \sim \frac{\text{var}(x)}{m^\beta} \quad (1)$$

Then taking the logarithm of both sides of the previous equation, it allows to be written as [16]:

$$\log[\text{var}[x^{(m)}]] \sim \log[\text{var}(x)] - \beta \log(m) \quad (2)$$

Now if we plot *variance of x* versus m on a log-log graph, the result should be a straight line with a slope of β .

For the WSN traffic analysis, the above formula was used for both topologies, and two plots for each were generated. First the logarithm of the variance of the payload length was computed for 7 different aggregated average payload length windows of 10, 32, 100, 320, and 1000 packets. The corresponding plot is illustrated below in Figure 22. The treadline corresponding to this plot is: $y = -0.0207x + 1.5543$ which indicates that β parameter is equal to 0.0207 which according to the theory provided above suggests a very high self-similarity with $H = 0.9897$.

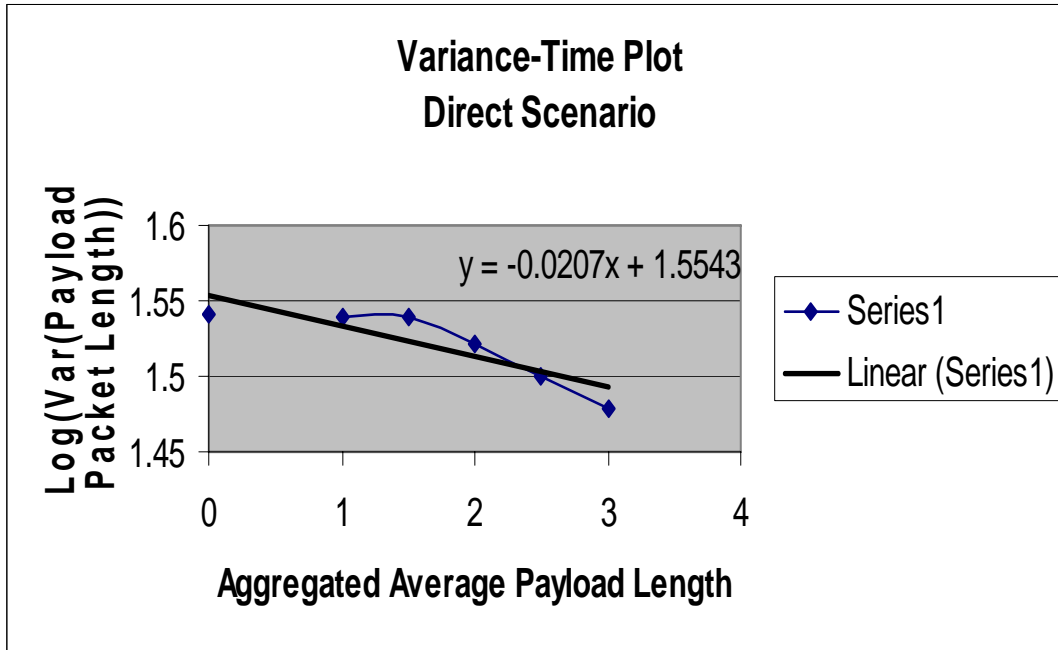


Figure 22. Logarithm of Variance Payload Length vs. Aggregated Average Payload Length Plot for the direct scenario.

Following the concept above the logarithm of the interarrival packet time versus the aggregated time is plotted in Figure 23. The β parameter was found equal to 0.1039 which means that H is equal to 0.948 which suggests high self-similarity.

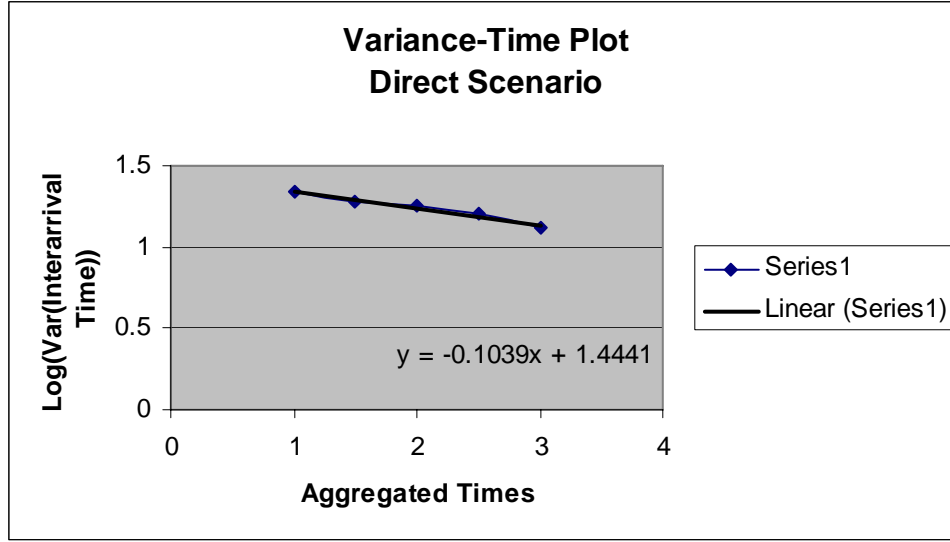


Figure 23. Logarithm of the Variance of packet Interarrival Time versus Aggregated Time plot for the direct to base communication scenario.

For the parent-child topology the same calculations as in the two paragraphs above were conducted, resulting in Figures 24 and 25. In Figure 24, the plot of the logarithm of the variance of the packet payload length versus aggregated average length is plotted that resulted in the following values for the β and H parameters: $\beta = 0.0207$ and $H = 0.9897$ values that suggest very long term time dependence.

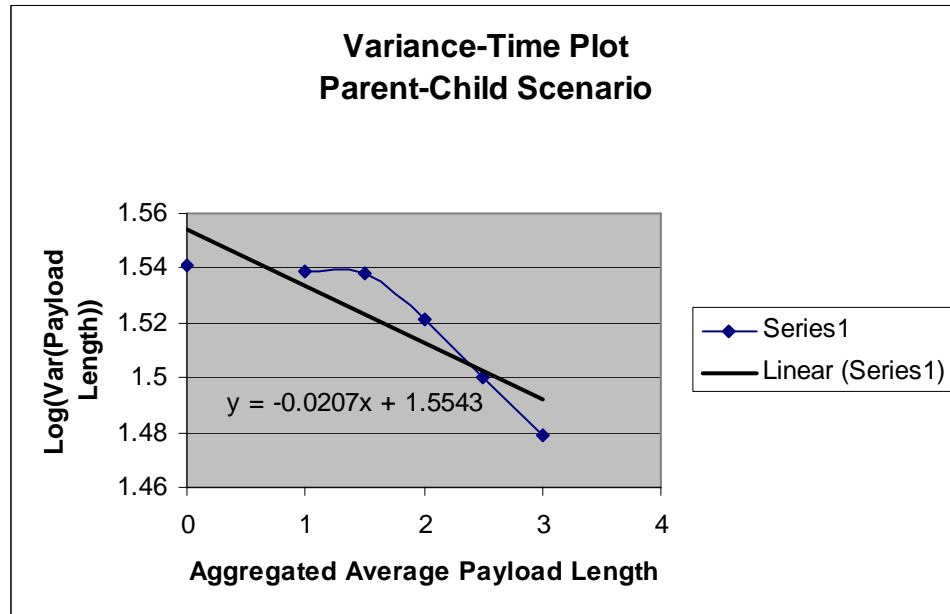


Figure 24. Logarithm of variance of packet Payload Length versus Aggregated Average Packet Length plot for the Parent-Child communication scenario.

Following the steps above the logarithm of the variance of the packet interarrival time versus aggregated time is illustrated in Figure 25 where from the slope of the treadline we see that β is equal to 0.2218 and so $H = 0.8891$. These values indicate high self-similarity.

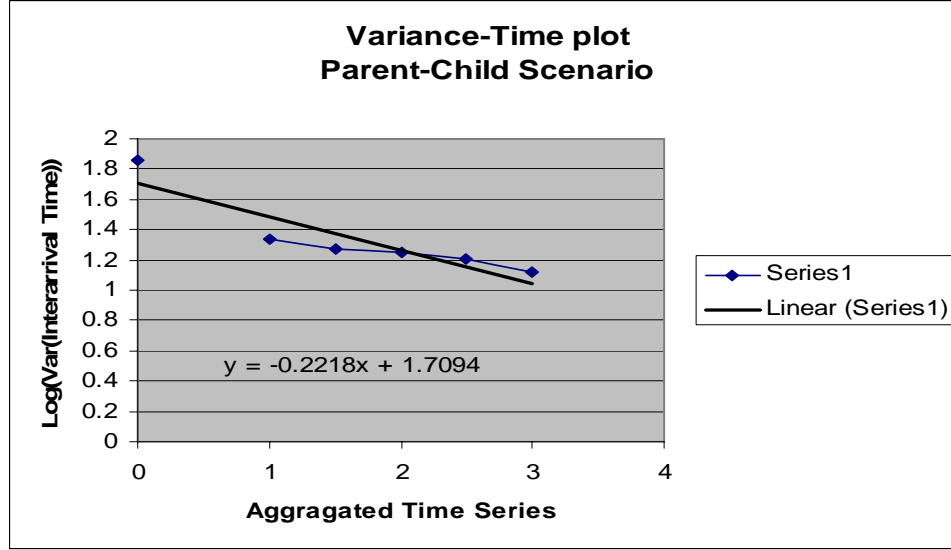


Figure 25. Logarithm of the Variance of packet Interarrival Time versus Aggregated Time plot for the Parent-Child communication scenario.

C. SUMMARY

Summarizing, it was shown that in a wireless sensor network the active message index identifies different traffic type and can be successfully used as one way to categorize traffic. The three type of traffic observed were Data, Routing (Routing Update Message and Health packets), and Acknowledgements each of which had different payload lengths, originating and destination addresses. Those differences were explicit and any variation of these would imply a network attack. Next using the logarithm of the variance of payload length and interarrival time for different time and length aggregations showed that this traffic entailed self-similarity properties. It should be noted that the sample size of collected packets was relatively small, potentially skewing the results, however for the purposes of this thesis, they provide a good initial indicator.

The percentage of each traffic type was obtained, which found that data traffic encompassed 71% of the total, health and routing 23%, and finally Acknowledgement 6%.

THIS PAGE INTENTIONALLY LEFT BLANK

VII. CONCLUSIONS AND FUTURE WORK

The human need for real time information on the physical environment has driven the evolution of many technologies and applications. The wireless sensor network satisfies that need, and the technology is growing exponentially. With sizes of less than one half of a hand or sometimes one tenth of a penny, these devices provide a large variety of information. Their applications extend from military to agricultural, from civil engineering to health monitoring, and from catastrophe detection to home intelligence. In areas that are not accessible or where there is no IT infrastructure, these networks provide an inexpensive, easy to use and install solution for remote sensing, in real time.

A. CONCLUSIONS

Traffic capture and analysis on a wireless sensor network was performed successfully and the statistical profiling that was performed revealed important information. The traffic was characterized according to the Active Message (AM) index, an index that indicates the handling of each message to the appropriate application. Four different AM indexes were observed identifying Data, Routing, Health and Acknowledgement types of traffic. Specific payload lengths, originating and destination addresses, and interarrival times were identified for each packet type. Those traffic statistics and characteristics were capable of producing a baseline for normal behavior. The latter along with other parameters when compared with real time traffic patterns can alert for traffic anomalies, and so can be used as an intrusion detection tool.

The examination of the packet length and the interarrival time of the packets in different time aggregations, in a variance-time plot on logarithmic scale, indicated that the traffic is self-similar. That was also witnessed by visual examination of the traffic, where each packet type was clustered. This means that every packet is largely dependent on the previous packets received, and upon reception of a specific message type, the same packet type is more likely to be received.

A comparison of the different packet type payload lengths and interarrival times exposed some discrepancies with the protocol specifications as they were explained in Chapter V. The payload lengths observed ranged from 11 to 27 bytes and never exceeded

the 29 bytes, which was the maximum packet length for the protocol used. The Health messages have the same properties as the Routing update messages, and the interarrival time expected for low power radio transmission was 60 seconds. For the first direct scenario this was calculated to be around to 11.735 seconds, which introduces a large variation. A similar variation was found for the parent-child scenario with interarrival times 7.41 and 18.65 seconds for Node 1 and 2 respectively. Moving to the data messages, interarrival times were expected to be 10 seconds for high power transmissions and for the direct scenario was calculated 2.046 seconds. For the parent-child scenario, interarrival times were 2.861 and 1.329 seconds for Node 1 and 2 respectively. The routing updates message interarrival time was expected to be 60 seconds for low power and was measured, for the direct scenario, to be 46.969 and 41.082 seconds for each mote that introduced a small variation in comparison to the parent-child scenario. In the latter scenario the interarrival time was calculated to be 18.5495 and 19.266 seconds for the two motes, respectively.

B. FUTURE WORK

Several topics are recommended for further research and are summarized as follows.

A program that will generate alerts, based on the traffic anomalies produced with the comparison of normal to real time traffic, should be written and be added to the *Sniffer* program. Upon completion of this program, a functional intrusion detection system will be available.

As seen in Chapter V, the CRC part of the packet was missing. The CRC is very important for the examination of modification of the message, and an investigation on why this part is not displayed and how this information can be captured should be performed. Also the algorithm that is used for the payload part of the packet should be retrieved, and the payload info should be used along with the rest packet information for the traffic profiling.

In this thesis, a total of three motes were used for traffic capturing, where a significant variation on the traffic characteristics was observed between direct and parent-child topologies. Traffic capturing and profiling should be performed in a larger network

consisting of 20 or more motes, deployed to cover a larger geographical area. Also two packet sniffers or more should be used for traffic capturing and then a cross comparison of the traffic captured by each should take place.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A.

A. SNIFFER JAVA CLASS

This is the main class of the *Sniffer* program [2]. The highlighted with blue color are the lines added for this thesis.

```
/* Sniffer.java
 * Main class for Sensor Network Sniffer.
 * Copyright (c) 2006 Hong-Siang Teo, Naval Postgraduate School (hteo@nps.edu).
 * All rights reserved.*/

import java.util.Iterator;
import java.util.LinkedList;
import java.util.StringTokenizer; //added categorization
import org.apache.commons.cli.*;
import java.io.*; //added

public class Sniffer {
    /*
     * Attributes and Options.
     */
    static String hostValue = "127.0.0.1";
    static int portValue = 9001;
    public static long packetCount = 0;
    public static long errorCount = 0;
    static Source source = null;
    static LinkedList<Output> outputsList = new LinkedList<Output> ();

    /**
     * @param args      Command line arguments
     */
    public static void main(String[] args) {

        /*Copyright (c) 2006 Georgios Kirykos, Naval Postgraduate School
        (gkirykos@nps.edu).
        * All rights reserved.*/

        //*****

        outputtoCsv oscv = new outputtoCsv();//added for traffic categorization
        displayWindow dw = new displayWindow();//added for traffic categorization
```

```

//*****
*****
/*
 * The main work flow is as follows:
 * - Parse arguments
 * - Instantiates a Packer Server to read packets from
 * - Perform protocol processing
 * - Perform application processing
 * - Outputs to terminal, or file if -o is specified
 */

// Parse arguments
parseArgs (args);

// Instantiates a new source, if it has not been instantiated already.
// The default source is Serial Forwarder.
if (source == null) {
    source = new SFSource (hostValue,portValue);
}

// Instantiates a Packet Server
PacketServer packetServer = new PacketServer (source);
if (packetServer.start() == false) {    // try to start the packet server
    System.out.println("Error starting packet server");
    System.exit(1);
}

// Initialize outputs. ScreenOutput is always available.
Writer.addOutput (new ScreenOutput ());
try {
    Writer.open();
} catch (Exception e) {
    System.err.println(e.getMessage());
    try {
        Writer.close();
    } catch (Exception e1) {}
}

// Get and process packets
for (;;) {
    // Get packet
    Packet packet = packetServer.getPacket();
    packetCount++;
}

```

```

        // Process packet
        try {
            Protocol.process (packet);
            AM.process (packet);

/***** Modified for Traffic Categorization August 28, 2006*****/
/*Copyright (c) 2006 Georgios Kirykos, Naval Postgraduate School (gkirykos@nps.edu).
 * All rights reserved.*/

String tempPacket = ScreenOutput.generateOutputString (packet);
StringTokenizer st = new StringTokenizer(tempPacket," ");

        System.out.println("Packet Categorization");
        String packetTime = st.nextToken(); //get time
        st.nextToken(); //ignore TOS
        String packetIDAM = st.nextToken(); //get GroupID, Destination and AM
        String numBytes = st.nextToken(); //get number of bytes
        String numberOfBytes = numBytes.substring(1,3);//get the number without the
        parenthesis of# bytes
        String packetData = st.nextToken(); //get data part
        String packetNodeID = packetData.substring(1,3); //get the first two data point
        that identify node that packet was sent from

        st = new StringTokenizer(packetIDAM,"," ); //chop down Group ID field

        String packetGroupID = st.nextToken(); //get Group ID
        String packetDestination = st.nextToken(); // destination address
        String packetAM = st.nextToken(); // get AM number

/*Put it in a format with commas for CSV process*****/

System.out.println("Group ID - Destination - AM - Time - Node ID - Number of
Bytes" );
System.out.println(packetGroupID + " , " + packetDestination + " , " + packetAM
+ " , " + packetTime+ " , " + packetNodeID);

        oscv.printTraffic(packetGroupID , packetDestination , packetAM , packetTime ,
        packetNodeID , numberOfBytes);//Puts traffic elements into a CSV file
        dw.displayTraffic(packetGroupID , packetDestination , packetAM , packetTime ,
        packetNodeID , numberOfBytes);//Display traffic window

/***** End Modification for Traffic categorization *****/

```

```

        catch (Exception e) {
            System.err.println(e.getMessage());
        }
    }

    public void displayTraffic(String traffic)
    {
    }

    private static void parseArgs (String[] args) {
        /* Parse arguments. */
        CommandLine cmd = null;
        Options options = new Options ();
        // Define options
        options.addOption (OptionBuilder.withArgName ("host")
        .hasArg ()
        .withDescription ("The host running Serial Forwarder. " +
        "Can be a valid hostname or IP address. " + "Default is local host.")
        .create("h"));
        options.addOption (OptionBuilder.withArgName ("port")
        .hasArg ()
        .withDescription ("Network port used by Serial Forwarder. " + "Default is 9001.")
        .create("p"));
        options.addOption (OptionBuilder.withArgName ("filename")
        .hasArg ()
        .withDescription ("Output to file in CSV format.")
        .create("oc"));
        options.addOption (OptionBuilder.withArgName ("filename")
        .hasArg ()
        .withDescription ("Output to file in raw binary format.")
        .create("ob"));
        options.addOption (OptionBuilder.withArgName ("filename")
        .hasArg ()
        .withDescription ("Log screen output to file.")
        .create("l"));
        options.addOption (OptionBuilder.withArgName ("mica|micaz|telos")
        .hasArg ()
        .withDescription ("Specifies TOS message format when
auto-detection fails.")
        .create("m"));
        options.addOption ("?", "false", "Print this help text.");

        // Parse command line options
        CommandLineParser parse = new BasicParser();
        try {
            cmd = parse.parse(options,args);

```

```

    } catch (ParseException e) {
        cmd = null;
    }
    if ((cmd == null) || (cmd.hasOption("?"))) {
        HelpFormatter f = new HelpFormatter();
        f.printHelp("java Sniffer,"options,true);
        if (cmd==null) {
            System.exit (1);
        } else {
            System.exit (0);
        }
    }
    // Perform some sanity checks, and set system settings based on options
    if (cmd.hasOption("h")) {
        hostValue = cmd.getOptionValue("h");
        // TODO: Sanity check on the host value?
    }

    if (cmd.hasOption("p")) {
        String portStr = cmd.getOptionValue("p");
        try {
            portValue = Integer.parseInt(portStr);
        } catch (NumberFormatException e) {
            System.out.println ("Port argument must be an integer.
Exiting.");
            System.exit(1);
        }
    }
    if ((portValue<0) || portValue>65535) {
        System.out.println ("Invalid port argument. Exiting.");
        System.exit (1);
    }
}
if (cmd.hasOption("oc")) {
    String filename = cmd.getOptionValue("oc");
    Writer.addOutput (new CSVFileOutput (filename));
}
if (cmd.hasOption("ob")) {
    String filename = cmd.getOptionValue("ob");
    Writer.addOutput (new BinaryFileOutput (filename));
}
if (cmd.hasOption("l")) {
    String filename = cmd.getOptionValue("l");
    Writer.addOutput (new LogFileOutput (filename));
}
if (cmd.hasOption("m")) {
    String value = cmd.getOptionValue("m");
    if (value.equalsIgnoreCase("mica")) {

```

```

        Packet.msgFormat= Packet.MSGFORMAT_MICA;
    } else if (value.equalsIgnoreCase("micaz")) {
        Packet.msgFormat = Packet.MSGFORMAT_MICAZ;
    } else if (value.equalsIgnoreCase("telos")) {
        // telos appears to use the same format as MICAZ
        Packet.msgFormat = Packet.MSGFORMAT_MICAZ;

    } else {
        Packet.msgFormat = Packet.MSGFORMAT_UNKNOWN;
    }
}
}
}
}

```

B. ACTIVE MESSAGE JAVA CLASS

This java program was created to parse the components of the packets in such way, so that all information of the packet will be ready for further analysis.

```

/* Added For packet categorization
 * activeMessage.java
 * Used to extract the AM value as well as: destination address, originating address,
group ID, time
 * Copyright (c) 2006 Georgios Kirykos, Naval Postgraduate School
(gkirykos@nps.edu).
 * All rights reserved.
 */

```

```
import java.util.StringTokenizer;
```

```

public class activeMessage {
    public void writeFile (Packet packet)
    {
        String line_item = new String();
        StringTokenizer st = new StringTokenizer(packet.toString(), " ");
        while (st.hasMoreElements())
        {
            line_item = st.nextToken();
            System.out.println(line_item);
        }
    }
}

```

C. OUTPUTTOCSV JAVA CLASS

This java class put in to a CSV file, values separated with comma.

```
/*Added For packet categorization
 * outputtoCsv.java
 * Used to map the AM value as well as: destination address, originating address, group
ID, time
 * to a CSV file (so then can be imported to an excel file) for further analysis and process
 * Copyright (c) 2006 Georgios Kirykos, Naval Postgraduate School
(gkirykos@nps.edu).
 * All rights reserved.
 */
```

```
import java.io.*;
//import javax.swing.*;
class outputtoCsv {
    int count=0;
    String Filename="c:\\outputtocvs_parentChild.txt";
    public void printTraffic (String packetGroupID,String
packetDestination,String packetAM,String packetTime,String packetNodeID, String
numberOfBytes )
    {
        count ++;
        if (count == 60000)
        {
            Filename+="1";
        }
        File cvs = new File(Filename);
        try
        {
            BufferedWriter cvsout = new BufferedWriter(new FileWriter(cvs,true));
            cvsout.write(packetGroupID + " , " + packetDestination + " , " +
packetAM + " , " + packetTime+ " , " + packetNodeID + " , " + numberOfBytes);
            cvsout.newLine();
            cvsout.close();
        }
        catch (IOException e)
        {
        }
    }
}
```

D. DISPLAYWINDOW JAVA CLASS

Provides a GUI for the traffic elements.

```
/*
Added For packet categorization
 * displayWindow.java
```

```

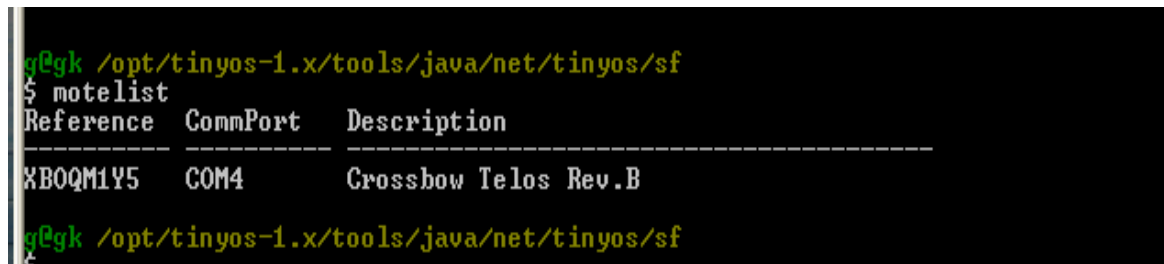
    * Used to display the AM value as well as: destination address, originating address,
    group ID, time
    * Copyright (c) 2006 Georgios Kirykos, Naval Postgraduate School
    (gkirykos@nps.edu).
    * All rights reserved.
    */
import javax.swing.*;
class displayWindow {
    JFrame jf = new JFrame("Traffic Categorization");
    JTextArea out = new JTextArea("Group ID - Destination - AM -      Time -
Node ID - Number of Bytes ");
    public displayWindow() {
        jf.add(out);
        jf.setSize(600,500);
        jf.setLocation(300, 150);
        out.setVisible(true);
        jf.setVisible(true);
    }
    public void displayTraffic (String packetGroupID,String
packetDestination,String packetAM,String packetTime,String packetNodeID,String
numberOfBytes ) {
        String temp;
        temp = out.getText() + "\n";
        temp += packetGroupID + "      ,      " + packetDestination + "      ,
" + packetAM + "      ,      " + packetTime+ "      ,      " + packetNodeID + "      ,
+ numberOfBytes;
        out.setText(temp);
    }
}

```

APPENDIX B.

The scope for this appendix is to provide the user with some basic setting needed to configure the Serial Forwarder. All information needed to set up the WSN, in terms of installing the TinyOS, program the micaz's and the TelosB can be found in reference [1].

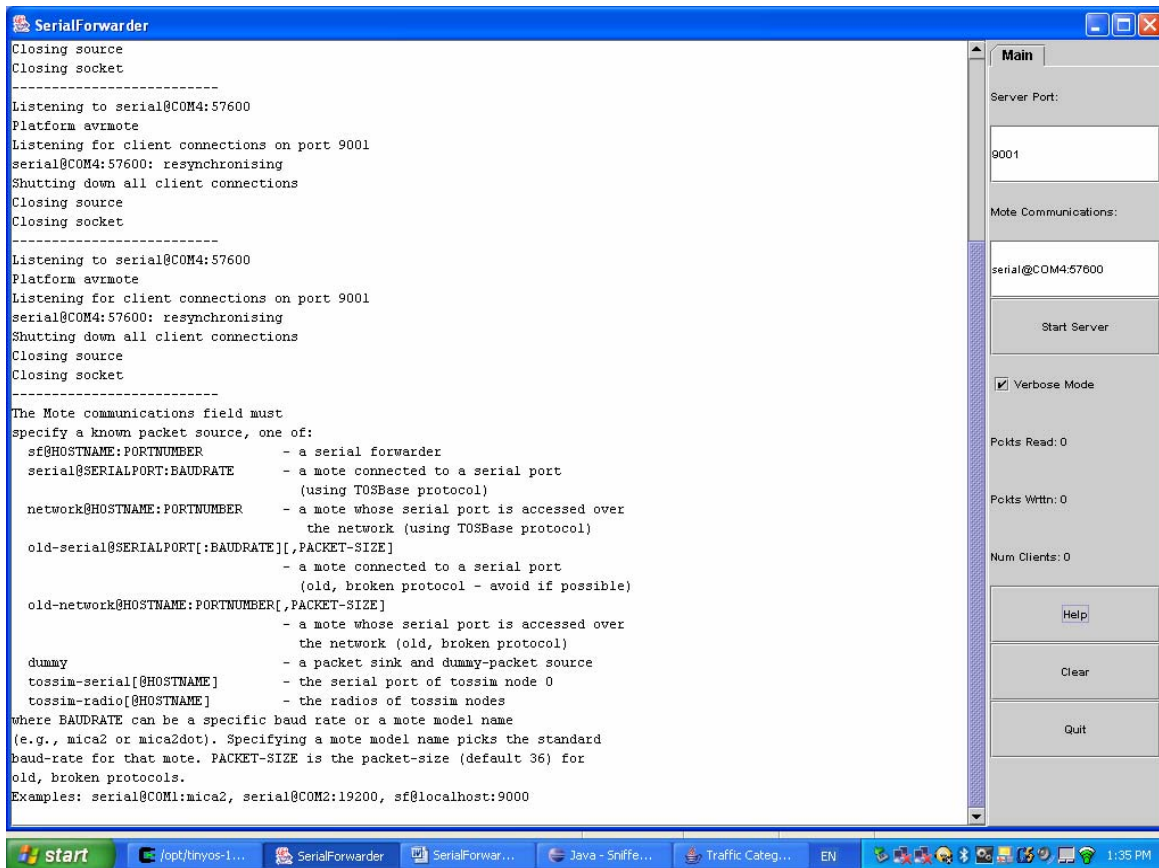
In order to determine the USB port that the TelosB is connected to the PC that is used for the packet capturing the user should open cygwin window and type: motelist. The outcome is printed below and consists of three columns, where the second one with Heading CommPort identifies the comm. That the TelosB is connected:



```
g@gk /opt/tinyos-1.x/tools/java/net/tinyos/sf
$ motelist
Reference  CommPort  Description
-----
XBQM1Y5    COM4       Crossbow Telos Rev.B
g@gk /opt/tinyos-1.x/tools/java/net/tinyos/sf
$
```

Reference	CommPort	Description
XBQM1Y5	COM4	Crossbow Telos Rev.B

Moving to the Serial Forwarder setting, the user should stop the server and enter the following settings: Communication port and baud rate, which for this experiment where respectfully com4 and 57600. A help button is also available and its display as well the above settings are illustrated in the following figure:



LIST OF REFERENCES

1. *Crossbow Technology Inc.*, “Wireless Sensor Networks Seminar,” San Jose, February 7-9, 2006.
2. Ethereal, <http://www.ethereal.com/introduction.html>. Last accessed November 2006.
3. Tcpdump, http://www.tcpdump.org/tcpdump_man.html. Last accessed November 2006.
4. Hong-Siang Teo, “Security of Sensor Networks,” Master’s Thesis, Naval Postgraduate School, June 2006.
5. Mohammad Ilyas and Imad Mahgoub, “Handbook of Sensor Networks: Compact Wireless and Wired Sensing Systems,” CRC Press, 2005.
6. SECON, <http://www.ieee-secon.org>. Last accessed November 2006.
7. Feng Zhao and Leonidas Guibas, “Wireless Sensor Networks, An Information Processing Approach,” Morgan Kaufmann, 2004.
8. Wikipedia, <http://www.wikipedia.com/wiki/MEMS>. Last accessed October 2006.
9. SUMMiTTM Technologies, www.mems.sandia.gov . Last accessed October 2006.
10. MEMS and Nanotechnology Exchange Inc., www.mems-exchange.org/mems/. Last accessed in October 2006.
11. TinyOS, <http://www.tinyos.net>. Last accessed October 2006.
12. Crossbow Technology Inc., “Xmesh User’s Manual,” March 2006.
13. TinyOS, <http://www.tinyos.net/tinyos-x/doc/serialcomm/description.html>. Last accessed December 2006.
14. TinyOS, <http://www.tinyos.net/tinyos-1.x/doc/tutorial/lesson6.html>. Last accessed December 2006.
15. Phil Porras, Dan Schnackenberg, Stuart Staniford-Chen, Maureen Stillman and Felix Wu, “The Common Intrusion Detection Framework Architecture,” <http://gost.isi.edu/cidf/drafts/communication.txt>. Last accessed November 2006.
16. William Stallings, “High-Speed networks and Internets,” 2nd ed. Prentice Hall, 1996, ch. 9.

17. William Stallings, "Wireless Communications and Networks," 2nd ed., Pearson, 2005.
18. Crossbow Inc., <http://www.xbow.com>. Last accessed October 2006.
19. Berkeley University of California, <http://www.coe.berkeley.edu/archive/users/warneke-brett/SmartDust/index.html>. Last accessed October 2006.
20. Smartdust Inc., <http://www.smartdust.com>. Last accessed October 2006.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
Ft. Belvoir, Virginia
2. Dudley Knox Library
Naval Postgraduate School
Monterey, California
3. Chairman
Department of Electrical and Computer Engineering
Naval Postgraduate School
Monterey, California
4. Chairman
Department of Physics
Naval Postgraduate School
Monterey, California
5. Professor John McEachen
Department of Electrical and Computer Engineering
Naval Postgraduate School
Monterey, California
6. Professor Murali Tummala
Department of Physics
Naval Postgraduate School
Monterey, California
7. Embassy of Greece, Naval Attaché
Washington DC
8. LT Georgios Kirykos
Hellenic Navy General Staff
Athens, Greece